

CHIAPAS EDI ENTERPRISE USER MANUAL

© Copyright 2015 Chiapas EDI Technologies, Inc. All rights reserved. This work may not be copied or distributed without written prior authorization.

This company and product have no affiliation with Mexico or the Mexican State of Chiapas.



Contents

INTRODUCTION	1
WHO IS THIS PRODUCT FOR?	2
INSTALLING CHIAPAS EDI ENTERPRISE	3
REQUIREMENTS	3
INSTALLATION	3
UPDATES	4
QUICK-START GUIDE	5
DECODING	5
ENCODING	9
AUTOMATION	12
USER REFERENCE	17
What is Chiapas EDI Enterprise?	17
CHIAPAS GATE INTERMEDIATE FORMAT	18
GENERAL CONCEPTS	18
CGIF Flat	20
CGIFXML2 Format	24
CHIAPAS EDI ENTERPRISE Registers and Runtime Environment	26
CHIAPAS STUDIO INTERFACE	28
Register Information	28
Data I/O Pane	29
Flat/HKey Register Pane	36
Mapping Interface	37
SegPool Register Interface	38
Menu Options	38
WORKFLOW INTERFACE	41
SCHEDULER INTERFACE	45
TECHNICAL REFERENCE	50
CORE DATABASE SCHEMA	51
AUTOMATION STORAGE SCHEMA	57
Flat BIN Storage	58
XML Storage	59
SAMPLE OBJECT SCHEMA	96
SEED TRANSACTIONS	103
SEED FILES	103

ETL Export Views	103
WORKFLOW ACTIVITIES	105
CGateTransfer	106
CGateMsg.....	110
CGateAckParse	112
CGateAckGenerate.....	113
CGateAckTA1Generate	113
CGateSeg	114
CGateHKeyFromFlat.....	114
CGateHKeyToFlat	114
CGateHKeyFromXMLDB	115
CGateHKeyToXMLDB	116
CGateHKeyToXElement.....	116
CGateXltEncode	117
CGateXltDecode	119
CGateCNNAdd.....	120
CGateCNNOpen.....	120
CGateCNNClose.....	121
CGateCNNSQLExec.....	121
CGateCNNSQLExecScript.....	122
CGateCNNSQLGetScalar.....	122
CGateCNNFlatToTable	123
CGateCNNFlatFromTable	123
CGateCNNFetchDT	124
CGateCNNPutDT	124
CGateDTToFlat	125
CGateDTFromFlat.....	125
CGateHKeyToXML	125
CGateHKeyFromXML.....	125
CGateQueryState	126
CGateClearRegister	130
CGateArg	130
CGateMerge	131
CGateForceMerge	131
CGateXFEROpenSession.....	132

CGateXFERCloseSession.....	132
CGateXFERPutFile	132
CGateXFERGetFile	133
CGateXFERDeleteFile	133
CGateXFERGetDirectoryFiles.....	133
CGateSendMail	134
CGateIOCopyFile	136
CGateIODeleteFile.....	136
CGateIODirectory	136
CGateRunShell.....	136
CGateRunShell2.....	137
SOLUTION WORKBOOK.....	138
SOLUTION: Legacy Data to 837P Claim EDI Files	138
Business Requirements.....	138
Technical Analysis	140
Technical Implementation	142

INTRODUCTION

Chiapas EDI Enterprise (pronunciation: Ch-i-A-pas) is an integration middleware platform focused on healthcare. Specifically, it focuses on the EDI transaction sets that HIPAA law mandates the way enterprises can exchange healthcare information – professional and institutional billing, enrollment, eligibility, remittances and so on. Just as healthcare is a complex business undertaking here in the United States, this is reflected in the EDI transactions sets – some of them requiring hundreds of pages to define.

The technology underlying Chiapas EDI began development in 2002 and was licensed commercially in 2003 as Chiapas EDI Version 1. In 2005, this product was completely open-sourced where today it has an active following. In 2010, a new company was formed around a complete rewrite of the core technology, and the Chiapas EDI 2012 product was the result. The main design goal of this product was to make these transactions as accessible as possible to enterprises at all levels, and judging by the solid demand for this product and the ability of our customers to quickly meet their business requirements, we are happy to say this goal was completely met. Chiapas EDI's ability to easily transform between EDI and database tables made it very straightforward to implement new EDI trading partners.

For Chiapas EDI Enterprise, we are committed to improving all areas of previous versions according to customer feedback and requests. A particular focus for this release are the enterprise users who approach EDI as a business intelligence data source as opposed to an operation data feed. To summarize the improvements we've made for this new release:

Improved Data Analytics capabilities

- The new XML encoder and decoder engines allow a completely new development path for working with EDI transactions that lends itself well to large scale analytics and reporting. For enterprises dealing with hundreds of different trading partners and many different EDI formats, XML translation capability offers distinct benefits when compared to the Chiapas EDI 2012 flat SQL table translation path.

Completely new 'out-of-the-box' capabilities

- Chiapas EDI Enterprise has over a dozen workflows that are running as soon as the platform is installed. These workflows are based off of Chiapas EDI's custom extensions to Microsoft's Windows Workflow Foundation 4 technology, so they are ready to be tailored at will.
- A completely new implementation of the 'BIN' system means that Chiapas EDI Enterprise now stores data for all supported transactions.
- Chiapas EDI now has the capability to store 'lossless' representations of the original EDI file in the BIN system

Improved Support Options

- Every license of Chiapas EDI 2012 or Chiapas EDI Enterprise now comes with a number of hours of direct phone and development support in addition to our next-business day Email Service Level Agreement so that our clients are assured they have a fully supported solution.

Improved Documentation

- Probably the most significant and tangible improvement over Chiapas EDI 2012 is a heavy emphasis on real-world examples in the documentation so that developers and analysts get a clear picture of what they need to do, and how to get there.

Given all of the feedback from our clients as well as the powerful new features in this release, we feel confident that healthcare business communications has never been easier than it is now.

WHO IS THIS PRODUCT FOR?

Chiapas EDI Enterprise is oriented towards MIS professionals and analysts who have some experience with core EDI concepts, such as outer and inner envelopes, segments, elements and so on. Healthcare experience and knowledge of important concepts such as HIPAA compliance, encryption requirements, safeguarding Protected Healthcare Information are very important for everyone who works with healthcare information.

Developers and analysts working with Chiapas EDI Enterprise should have basic knowledge of how to install and set up Microsoft SQL Server, versions 2008 and above. If one wants to customize the default workflows, then a knowledge of Windows Workflow Foundation 4 as well as the .NET 4.0 environment is important.

Finally, it is critical to have access to properly licensed HIPAA Implementation Guides, as published by the Washington Publishing Company (wpc-edi.com). They are the sole publishers on behalf of the national standards body, ASC X12. Even though an enterprise may receive a companion guide from a trading partner, it should be explicitly understood that all such companion guides are *supplements* to the core HIPAA Implementation Guides, and will not contain the information necessary to properly handle HIPAA EDI transactions.

INSTALLING CHIAPAS EDI ENTERPRISE

REQUIREMENTS

Chiapas EDI Enterprise requires the following minimum hardware requirements:

- 64-Bit Windows Operating System: Windows 7, Windows Server 2008 R2, Windows 8 or Windows Server 2012
- .NET 4.0. Although installed by default on Windows 7 and 8, it is normally an optional Feature on Server operating systems.
- SQL Server 2012 R2 or Above (Express or Standard versions)
- 8 GB of RAM
- 5GB Free disk space
- Local Administrator rights when installing

By default, Chiapas EDI installs to a fixed directory on the C: drive, C:\chiapas. This cannot be reconfigured, so it is important at least 5 GB is free on the C: drive. Although the default workflows are also tied to the C: drive, this is completely reconfigurable.

INSTALLATION

1. Chiapas EDI Enterprise is distributed as a single .MSI file. To run it, double-click on it and it will automatically install all components to the C: drive under the C:\chiapas directory. If the Evaluation Version is downloaded via the Chiapas EDI website, both Google Chrome and Internet Explorer will give warnings about running downloaded executable files – these warnings should be overridden. Note that **Chiapas EDI Enterprise is a 'silent' installer** and does not give any feedback if the install process is successful – this is normal.

2. Next, launch the database installer located at:

C:\chiapas\db\SQLPkgChiapasData.exe

Follow the instructions to add the chiapasData SQL database to the SQL Server instance. This contains all of the tables necessary for Chiapas EDI's automation system as well as the mock healthcare data and SQL Views for creating new outbound transactions.

3. The chiapasData database is crucial for the automation aspects of Chiapas EDI Enterprise. Chiapas EDI Enterprise uses the following file to locate this database when starting up:

C:\chiapas\bin\cnstr.txt

NOTE: If the destination server is not the local default instance, you will need to execute two steps:

- a. Update the connection string on the filesystem at C:\chiapas\bin\cnstr.txt
- b. Update the embedded connection string within the data shuttle trigger located within the chiapasData database. If you are running SQLEXPRESS, the following SQL will update the connection string:

```
UPDATE TSRC SET TSRC_DATA01 = 'Data
Source=.\SQLEXPRESS;Database=chiapasData;Trusted_Connection=true;' WHERE
TSRC_ID = 13
```

4. To configure and start the background automation service:
Press Start -> Right Click on Computer, select 'Manage', Go to Configuration, Services, Right-click Properties on ChiapasService. Switch to the LogOn tab, then fill in the 'Log On As' information with login and password information for an account with Administrator permissions.

Return to the General Tab, press 'Start' to start running the automation service.

UPDATES

Chiapas EDI Technologies, Inc. distributes the installer to registered clients via a Secure FTP server. It is necessary to have a SecureFTP client in order to access your specific account on the server; one is included with Chiapas EDI Enterprise within the C:\chiapas\bin folder called 'WinSCP'.

The site to access the server is:

`sftp.chiapas-edi.org`

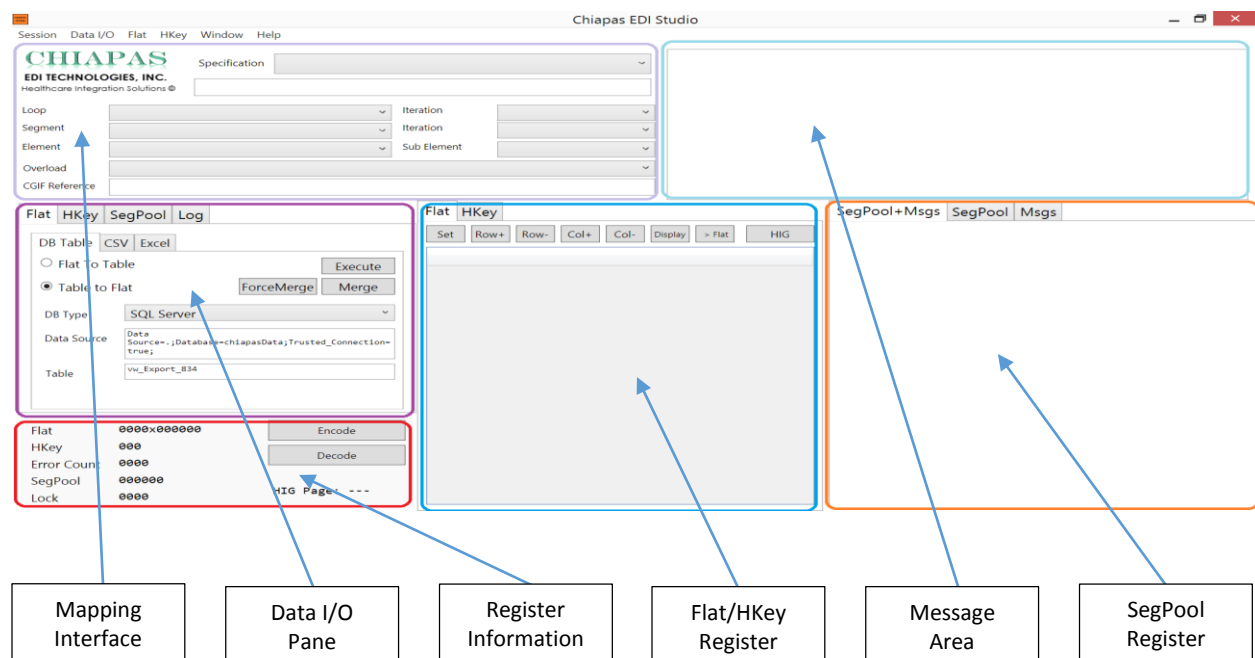
Login information is provided to you along with your Chiapas EDI Enterprise license.

QUICK-START GUIDE

This Quick Start Guide is oriented to quickly accessing three core features of the Chiapas EDI Enterprise product: Decoding, Encoding, and the built-in automation pipelines. Decoding is focused on taking one of the included sample HIPAA files through the steps of converting it to a database table. Encoding focuses on converting some of the included mock healthcare data to a new HIPAA file. This covers the most basic usage patterns of Chiapas EDI. Both of these examples start with launching Chiapas Studio, located at the following path:

`C:\chiapas\bin\ChiapasStudio.exe`

Before we proceed further, we should identify the functional areas of the ChiapasStudio interface:



A basic operation of Chiapas EDI 2012 is the ability to convert arbitrary 4010 or 5010 EDI files to database tables. This section will guide you through the process step by step.

1. Open ChiapasStudio (`C:\chiapas\bin\chiapasstudio.exe`; you may need to run as Administrator)
2. On the front screen, you should see the Data I/O Pane default to the Flat tab. Click the SegPool tab:



- ```

SegPool+Msgs SegPool Msgs
[0000001] ISA^00^.....01^.....*ZZ^SAMPL SENDI
[0000002] GS^BE^SENDER ID^RCVR ID^20120810^0914^1^X^005
[0000003] ST^834^1000000000^005010X220A1
[0000004] BGN^00^1000000000^20120810^0917^*****RX
[0000005] N1^P5^CHIARAS EDI^TECH^FI^273801950
[0000006] N1^IN^DOLLITTLE HOSPITAL^FI^123400001
[0000007] INS^Y^18^030^A^*****F
[0000008] REF^0F^987654321
[0000009] REF^1L^BASICHEALTHCAREPLAN
[0000010] REF^17^100001
[0000011] DTP^P^356^DB^20120101
[0000012] NM1^IL^1^BURGER^JOHN
[0000013] PER^IP^*HP^4155551200
[0000014] N3^123 A ST
[0000015] N4^SAN FRANCISCO^CA^94117
[0000016] EMG^DB^13800707^M
[0000017] INS^Y^18^030^A^*****F
[0000018] REF^0F^987654321
[0000019] REF^1L^BASICHEALTHCAREPLAN
[0000020] REF^17^200001
[0000021] DTP^P^356^DB^20120101
[0000022] NM1^IL^1^FILET^STEVE
[0000023] PER^IP^*HP^4155551200
[0000024] N3^222 B ST
[0000025] N4^SAN FRANCISCO^CA^94117
[0000026] EMG^DB^19700404^M
[0000027] INS^Y^18^030^A^*****F

```

- ```

Flat < HWay
Flat < HWay

-<CGIFXMK2Root_T1>
<OEISA>
-<S01>
<ISA02_AUTH_NFO_OVL_NO_AUTH_NFO>.....</ISA02_AUTH_NFO_OVL_NO_AUTH_NFO>
<ISA04_SEC_NFO_OVL_PASSWD>.....</ISA04_SEC_NFO_OVL_PASSWD>
<ISA06_SENDR_ID_OVL_MUTLY_DEFNID_ID><SAMPL_SENDRID..</ISA06_SENDR_ID_OVL_MUTLY_DEFNID_ID>
<ISA08_RECVR_ID_OVL_MUTLY_DEFNID_ID><SAMPL_RECVRID..</ISA08_RECVR_ID_OVL_MUTLY_DEFNID_ID>
<ISA09_DT>120810</ISA09_DT>
<ISA10_TM>0914</ISA10_TM>
<ISA11_REPTN_SEPRTR>~</ISA11_REPTN_SEPRTR>
<ISA12_VERSN_NR>00501</ISA12_VERSN_NR>
<ISA13_JCN>100000000</ISA13_JCN>
<ISA14_ACK_REQ>0</ISA14_ACK_REQ>
<ISA15_USG_IND>P</ISA15_USG_IND>
<ISA16_SUBELE_SEP>+</ISA16_SUBELE_SEP>
</S01>
-<GSHDR>
<S01>
<GS02_APP_SENDR_CD><SENDR_ID></GS02_APP_SENDR_CD>
<GS03_APP_RECVR_CD><RECVR_ID></GS03_APP_RECVR_CD>
<GS04_DT>20120810</GS04_DT>
<GS05_TM>0914</GS05_TM>
<GS06_GCN>1</GS06_GCN>
<GS08_VERS_ID_CD><GS05010X220A1></GS08_VERS_ID_CD>
</S01>
<STHDR>
<S01>
<ST02_CTRLNR>100000000</ST02_CTRLNR>
<ST03_IMPLMNTN_CONVNTN_REFNC>005010X220A1</ST03_IMPLMNTN_CONVNTN_REFNC>
</S01>
<S02>
<BG001_TS_PURPS_CD>00</BG001_TS_PURPS_CD>
<BG002_TS_REF_NR>100000000</BG002_TS_REF_NR>
<BG003_TS_CREATN_DT>20120810</BG003_TS_CREATN_DT>
<BG004_TS_CREATN_TM>0917</BG004_TS_CREATN_TM>
<BG008_ACTN_CD>RX</BG008_ACTN_CD>
</S02>
<L1000A>
<S01>
<N1002_PLAN_SPONSR_NM>CHIAPAS EDI TECH</N102_PLAN_SPONSR_NM>
<N104_SPONSR_ID_OVL_FED_TAX_ID_NR>273801950</N104_SPONSR_ID_OVL_FED_TAX_ID_NR>
</S01>
<L1000B>
<L1000B>
<S01>
<N107_INCD_NM>0000 TITLE HOSPITAL</N107_INCD_NM>

```

Press the Flat tab within the Flat/HKey Register pane, and the Flat register is displayed:

[illegible]

The business information present within the original EDI file has now been transformed into both CGIFXML2 format (via the HKey Register) and the CGIF Flat format within the Flat Register. These two formats represent interfaces by which enterprises can parse the business information within the original EDI files, and are further defined later in this manual. The CGIF Flat format is generally the most accessible. First, we'll need to press the 'Flat' tab of the Data I/O pane and then type 'TUTORIAL_834' in the Table text box:

Flat

HKey

SegPool

Log

DB Table

CSV

Excel

☒ Flat To Table

Execute

☐ Table to Flat

ForceMerge

Merge

DB Type

SQL Server

Data Source

Data Source=.;Database=chiapasData;Trusted_Connection=true;

Table

TUTORIAL_834

Next, ensure the connection string in 'Data Source' is correct. For this demo, the default connection string points to the chiapasData database that is set up as part of the installation instructions. Select the 'Flat to Table' radio button, and then press 'EXECUTE'.

Next, open up SQL Server Management Studio, open a new query on the chiapasData database, and run the following query:

```
EXEC SP_COLUMNS 'TUTORIAL_834'
```

The results show the columns that have been created in the new table, which should contain 23 rows of data:

COLUMN_NAME
T1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO
OEISA_S01_ISA04_SEC_NFO_OVL_PASSWD
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID
OEISA_S01_ISA09_DT
OEISA_S01_ISA10_TM
OEISA_S01_ISA11_REPTN_SEPRTR
OEISA_S01_ISA12_VERSN_NR
OEISA_S01_ISA13_ICN
OEISA_S01_ISA14_ACK_REQ
OEISA_S01_ISA15_USG_IND
OEISA_S01_ISA16_SUBELE_SEP
OEISA_S03_IEA01_GS_CT
OEISA_S03_IEA02_ICN
GSHDR_S01_GS02_APP_SENDR_CD
GSHDR_S01_GS03_APP_RECVR_CD
GSHDR_S01_GS04_DT
GSHDR_S01_GS05_TM
GSHDR_S01_GS06_GCN
GSHDR_S01_GS08_VERS_ID_CD
GSHDR_S03_GE01_TS_CT
GSHDR_S03_GE02_GCN
STHDR_S01_ST02_CONTRL_NR
STHDR_S01_ST03_IMPLMNTN_CONVNTN_REFNC
STHDR_S02_BGN01_TS_PURPS_CD
STHDR_S02_BGN02_TS_REF_NR
STHDR_S02_BGN03_TS_CREATN_DT
STHDR_S02_BGN04_TS_CREATN_TIM
STHDR_S02_BGN08_ACTN_CD
STHDR_S10_SE01_SEG_CT
STHDR_S10_SE02_TCN
L1000A_S01_N102_PLAN_SPONSR_NM
L1000A_S01_N104_SPONSR_ID_OVL_FED_TAX_ID_NR
L1000B_S01_N102_INSR_NM
L1000B_S01_N104_INSR_ID_CD_OVL_FED_TAX_ID_NR
L2000_S01_INS01_MEM_IND
L2000_S01_INS02_INDVDL_REL_CD
L2000_S01_INS03_MAINTNC_TYPE_CD
L2000_S01_INS05_BENFT_STATS_CD
L2000_S01_INS08_EMPLYMNT_STATS_CD
L2000_S02_REF02_SUB_ID_OVL_SUB_NR
L2000_S03_REF02_MEM_GRP_POLCY_NR_OVL_GRP_POLCY_NR
L2000_S04_REF02_SUPP_ID_OVL_CLIENT_REPRTNG_CATGRY
L2000_S05_DTP03_STATS_NFO_EFF_DT_OVL_ELIG_BEGN_D8
L2100A_IL_S01_NM103_MEM_LNAME_OVL_PERSN
L2100A_IL_S01_NM104_MEM_FNAME
L2100A_IL_S02_PER04_COMM_NR_OVL_HOM_PHON_NR
L2100A_IL_S03_N301_ADRS_LIN
L2100A_IL_S04_N401_CITY_NM
L2100A_IL_S04_N402_STAT_CD
L2100A_IL_S04_N403_MEM_POSTL_ZON_ZIP_CD
L2100A_IL_S05_DMG02_BIRTH_DT_OVL_DT_CCYYMMDD
L2100A_IL_S05_DMG03_GENDR_CD

This table represents all of the information necessary to re-create the original EDI file.

ENCODING

This example will demonstrate how to use Chiapas EDI to progress from some simple, mock healthcare information and transform it into a fully compliant EDI file. Let's start with examining some of the mock healthcare information. First, let's investigate the Member table. Using SQL Server Management Services (SSMS), connect to the chiapasData database and open a new query window. Run the following query:

```
SELECT TOP 10 * FROM SAMPL_MEMBER
```

Results		Messages										
	MEM_ID	FIRST_NM	LAST_NM	SSN_ID	BIRTH_DT	GENDER	ADD1	ADD2	CITY	STATE	ZIP_CD	PHONE_NR
1	1	JOHN	BURGER	987654320	1980-07-07	M	123 A ST	NULL	SAN FRANCISCO	CA	94117	4155551200
2	2	STEVE	FILET	987654321	1970-04-04	M	222 B ST	NULL	SAN FRANCISCO	CA	94117	4155551201
3	3	LAWRENCE	HAM	987654322	1950-04-04	M	987 C ST	NULL	SAN FRANCISCO	CA	94117	4155551202
4	4	JAKE	NOODLES	987654323	1960-03-03	M	654 D ST	NULL	SAN FRANCISCO	CA	94117	4155551203
5	5	ABIGAIL	STROGANOFF	987654324	1990-08-08	F	123 E ST	NULL	SAN FRANCISCO	CA	94117	4155551204
6	6	DELORES	PEAS	987654325	1970-04-04	F	222 F ST	NULL	SAN FRANCISCO	CA	94117	NULL
7	7	PENELOPE	HASHBROWNS	987654326	1960-03-03	F	333 G ST	NULL	SAN FRANCISCO	CA	94117	4155551205
8	8	EDWIN	PASTA	987654327	1980-10-10	M	444 H ST	NULL	SAN FRANCISCO	CA	94117	NULL
9	9	LARRY	SANDWICH	987654328	2000-01-01	M	555 I ST	NULL	SAN FRANCISCO	CA	94117	4155551206
10	10	NAOMI	BREAD	987654329	1950-04-04	F	123 J ST	NULL	SAN FRANCISCO	CA	94117	4155551207

Now, here's to view the roster of providers included in the mock data:

```
SELECT TOP 10 * FROM SAMPL_PROVIDER
```

Results		Messages										
	PROV_ID	TAX_ID	NPI_ID	COMPANY_NM	FIRST_NM	LAST_NM	ADD1	ADD2	CITY	STATE	ZIP_CD	CONTACT_NM
1	1	123400001	NULL	DOOLITTLE HOSPITAL	NULL	NULL	123 A ST	NULL	SAN FRANCISCO	CA	94117	NIXON
2	2	NULL	NULL	WINDING ROAD HMO	NULL	NULL	456 B ST	NULL	SAN FRANCISCO	CA	94117	FORD
3	3	NULL	NULL	BEST PRACTICE CLINIC	NULL	NULL	789 C ST	NULL	SAN FRANCISCO	CA	94117	CARTER
4	4	NULL	NULL	CHEAPEST HMO	NULL	NULL	199 D ST	NULL	SAN FRANCISCO	CA	94117	GRANT
5	5	273801950	NULL	CHIAPAS EDI TECH	NULL	NULL	126 CARDENAS AVE	NULL	SAN FRANCISCO	CA	94132	RICHARD TEMPS
6	6	273801950	123400001	NULL	LAWRENCE	AVOCADO	999 Z ST	NULL	SAN FRANCISCO	CA	94132	ELDRITCH POWERS

Now, to bring up the mock claims information:

```
SELECT * FROM SAMPL_CLAIM_HDR HDR INNER JOIN SAMPL_CLAIM_DTL DTL ON HDR.CLAIM_ID = DTL.CLAIM_ID
```

	CLAIM_ID	PROV_ID	MEM_ID	PATNT_ACCT_NR	FAC_TYP_CD	PRI_DIAG_CD	SEC_DIAG_CD	STATUS	ADJUDCTN_DT	CLAIM_DTL_ID
1	1	1	1	2012000000001	NULL	34600	NULL	PAID	2012-08-01	1
2	2	2	2	2012000000002	NULL	V700	NULL	PAID	2012-09-01	2
3	3	3	3	2012000000003	NULL	78900	NULL	DENIED	2012-08-01	3
4	4	4	4	2012000000004	NULL	34600	V061	DENIED	2012-08-01	4
5	5	1	5	2012000000005	NULL	4659	NULL	DENIED	2012-08-01	5
6	6	2	6	2012000000006	NULL	V741	NULL	PAID	2012-09-01	6
7	7	3	7	2012000000007	NULL	4659	NULL	DENIED	2012-09-01	7
8	8	4	8	2012000000008	NULL	6918	V061	PAID	2012-08-01	8
9	9	1	9	2012000000009	NULL	34600	NULL	PAID	2012-08-01	9
10	10	2	10	2012000000010	NULL	34600	NULL	DENIED	2012-09-01	10
11	11	3	11	2012000000011	NULL	78900	NULL	DENIED	2012-08-01	11
12	12	4	1	2012000000012	NULL	34600	NULL	PAID	2012-08-01	12

On the previous page, we saw member and provider information, and above we can see a mock claims table that will be encoded to a file. There is a single SQL View with the chiapasData database that can be used to send data directly to a new 837 Professional file, 'vw_Export_837P'. The difficulty with using the view directly to create EDI files is that SQL Server does not guarantee that the ORDER BY clauses will function correctly within Views. To address this, there is a stored procedure included within the chiapasData that will work to ensure that the output of any CGIF formatted table will be properly ordered and that claims, line items and so on will be grouped together correctly.

Here are the sequence of steps necessary to go from this sample export view to an EDI file:

1. Within Chiapas Studio, click on the 'Flat' tab on the left Data I/O pane, and then select the 'Table to Flat' radio button and type in EXEC USP_ORDEREDI 'vw_Export_837P' within the Table textbox.

The screenshot shows the 'Flat' tab in the Chiapas Studio interface. It has sub-tabs for 'Flat', 'HKey', 'SegPool', and 'Log'. Under the 'Flat' sub-tab, there are options for 'DB Table' (CSV, Excel) and two radio buttons: 'Flat To Table' and 'Table to Flat' (which is selected). There are buttons for 'Execute', 'ForceMerge', and 'Merge'. The 'DB Type' is set to 'SQL Server'. The 'Data Source' is 'Data Source=.;Database=chiapasData;Trusted_Connection=true;'. The 'Table' field contains 'EXEC USP_ORDEREDI 'vw_Export_837P''.

2. Press Execute.
3. Next, press the HKey tab on the Flat/HKey Register pane and then press the Flat > HKey button:

The screenshot shows the 'Flat/HKey Register' pane. It has tabs for 'Flat' and 'HKey'. The 'HKey' tab is selected. Below the tabs are two buttons: 'Flat < HKey' and 'Flat > HKey'. The 'Flat > HKey' button is highlighted with a blue circle.

4. This will effectively 'pull' the Flat Register into the HKey register. The HKey tab will show a representation of the HKey in XML form. Since the HKey is now loaded, we need to press the ENCODE button in the lower-left hand area of the screen to translate the HKey to the SegPool register:

The screenshot shows the HKey register interface. It displays a table with the following data:

Flat	0078x000019
HKey	000
Error Count	00000
SegPool	000000
Lock	0000

Below the table, there are two buttons: 'Encode' and 'Decode'. The 'Encode' button is highlighted with a blue circle. At the bottom right, it says 'HIG Page: ---'.

5. The SegPool Register will now be loaded with an EDI transaction. The only remaining step is to save the SegPool register to a new file via the Data I/O Pane. Click on the SegPool tab of the Data I/O pane, then the File sub-tab. Select the SegPool to File radio button, check the CR and LF check boxes to ensure the resulting file will have CR/LF formatting, enter in a filename and press EXECUTE:

The screenshot shows the 'Data I/O Pane' with the 'SegPool' tab selected. Within this tab, the 'File' sub-tab is active. The 'SegPool to File' radio button is selected, and the 'CR' and 'LF' checkboxes are checked. The 'Execute' button is highlighted. The 'SegPool File' text box contains the path 'C:\chiapas\tutorial_837p.txt'. The 'SegSep' is set to '~' and the 'EleSep' is set to '*'.

At this point, a new file called 'tutorial_837p.txt' will appear in the C:\chiapas directory. Starting from raw data located within the SAMPL_* tables of the chiapasData database, we loaded the raw data via a stored procedure that guarantee proper sort order and used the ChiapasStudio interface to travel the data to a HIPAA compliant EDI transaction. This concludes the encoding tutorial.

AUTOMATION

Chiapas EDI Enterprise includes twelve pipelines as part of the normal distribution. A *pipeline* in Chiapas EDI Enterprise is defined as a process that accepts data inputs and produces data output, and is capable of running concurrently. In Chiapas EDI Enterprise, there is a single service ('ChiapasService') that runs in the background and handles all orchestration by listening for *triggers*. When a trigger condition is met, it will insert a new row into the **EVENTS** table. With the base licensing tier of Chiapas EDI Enterprise, there are two WorkerProcess programs also running in the background that poll for new incoming EVENTS. When one is found, the WorkerProcess attempts to take ownership, and if successful, processes it. Chiapas EDI Enterprise uses Microsoft Windows Workflow Foundation 4.0 (WWF 4) with custom, proprietary extensions to make the features of the technology accessible via drag-and-drop interface dialogs that are stacked and turned into workflows. When the workflow finishes execution, the WorkerProcess automatically sends any notifications or error messages to the **MSGLOG** table and updates the event to complete.

In this way, the orchestration is cleanly separated from the pipeline execution. Should a WorkerProcess crash or hang, the ChiapasService will restart it. Stopping ChiapasService will also stop all WorkerProcesses. If the ChiapasService process is killed, the running WorkerProcesses will automatically exit.

The pipelines included with the distribution are located in C:\chiapas\pipeline as follows. All of these are active once the installation is complete and the ChiapasService is started:

Active Triggers			
All Triggers			
Pending Events			
TSRC_ID	TSRC_TYPE	TSRC_NAME	TSRC_WF_FILENAME
12	SCHEDULED_EVENT	Pipeline 012_EncodeFromBINDB	C:\chiapas\xaml\Pipeline_012_EncodeFromBINDB.xaml
11	SCHEDULED_EVENT	Pipeline 011_EncodeFromXMLDB	C:\chiapas\xaml\Pipeline_011_EncodeFromXMLDB.xaml
10	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 010_EncodeFromXML	C:\chiapas\xaml\Pipeline_010_EncodeFromXML.xaml
9	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 009_EncodeFromCSV	C:\chiapas\xaml\Pipeline_009_EncodeFromCSV.xaml
8	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 008_DecodeToBINDB_ForceMerge	C:\chiapas\xaml\Pipeline_008_DecodeToBINDB_ForceMerge.xaml
7	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 007_DecodeToBINDB_Merge	C:\chiapas\xaml\Pipeline_007_DecodeToBINDB_Merge.xaml
6	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 006_DecodeToXMLDB	C:\chiapas\xaml\Pipeline_006_DecodeToXMLDB.xaml
5	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 005_DecodeToXML	C:\chiapas\xaml\Pipeline_005_DecodeToXML.xaml
4	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 004_DecodeToCSV	C:\chiapas\xaml\Pipeline_004_DecodeToCSV.xaml
3	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 003_ScrubPHI	C:\chiapas\xaml\Pipeline_003_ScrubPHI.xaml
2	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 002_CullLF	C:\chiapas\xaml\Pipeline_002_CullLF.xaml
1	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline 001_SetLF	C:\chiapas\xaml\Pipeline_001_SetLF.xaml

This view is obtained from the Scheduler Interface within ChiapasStudio; to access it, select the pulldown 'Window' menu and select Scheduler Interface. The first ten triggers are *active*, meaning that the ChiapasService is actively polling stimuli to see if a new event should be launched. The last two are *passive*, meaning that they need to be started by directly inserting into the EVENTS table and not via ChiapasService directly.

Note that there is a directory naming convention – all directories within the pipeline start with two or three zero-padded number digits. When in a pipeline, it's clear which directory is the starting point. A brief overview of the purpose and function of the twelve pipelines follows.

001_SetLF

Directories

01_in_edt
02_out_edt
03_err_edt

EDI files that are dropped into the input directory will be reprocessed to the output so that every segment ends with carriage return and line feed characters, making them easily viewable within a text editor. Files with critical decode errors will be redirected to the Err directory.

002_CullLF

Directories

01_in_edt
02_out_edt
03_err_edt

EDI files dropped into the input directory will be reprocessed so that the output is stripped of CR/LFs, which is format most clearinghouses prefer. Files with critical decode errors will be redirected to the Err directory.

003_ScrubPHI

Directories

01_in_edt
02_out_edt
03_err_edt

EDI files dropped into the input directory will be decoded and re-encoded with *most* Protected Health Information stripped out and replaced with randomized data. This includes most dates and personal name and address information. The files will still need to be checked manually to verify that PHI is removed – never assume that this process makes a file safe to transmit on open networks.

004_DecodeToCSV

Directories

01_in_edt
02_out_csv
03_done_edt
04_err_edt

EDI files placed in the input directory will be decoded and transformed to comma delimited CGIF Flat CSV files in the output directory. The CSV files will have column headers and all data elements and column names will be surrounded by double quote characters.

005_DecodeToXML

Directories

01_in_edt
02_out_xml
03_done_edt
04_err_edt

Input files will be decoded and then transformed to XML files on the output directory.

006_DecodeToXMLDB

Directories

01_in_edt
02_done_edt
03_err_edt

Input files will be decoded and then imported to the chiapasData.dbo.BIN_XML destination table, which will contain both the CGIFXML2 formatted XML file as well as the EVENT_ID for reference.

007_DecodeToBINDB_Merge

Directories

01_in_edt
02_done_edt
03_err_edt

Input files will be decoded and placed into the BIN tables via the Merge activity. The BIN tables follow the format of the seed 5010 files; any fields present within the decoded file that are not present within the destination table will trigger an 'Unmerged Mapping' error message that identifies the missing data.

NOTE: This workflow uses a 'Delayed Write' functionality. Therefore, the best way to validate the data operation is fully complete is to check the FSTOR_PENDS table to ensure the data is successfully written.

008_DecodeToBINDB_ForceMerge

Directories

01_in_edt
02_done_edt
03_err_edt

Input files will be decoded and placed into the BIN tables via the ForceMerge activity. The BIN tables follow the format of the seed 5010 files; any fields present within the decoded file that are not present within the destination table will cause the new fields to be added to the destination tables, so that there is no data loss. Note that this is not enabled for concurrent execution due to schema locks placed on the database tables when the workflow runs; files will be processed one at a time.

NOTE: This workflow uses a 'Delayed Write' functionality. Therefore, the best way to validate the data operation is fully complete is to check the FSTOR_PENDS table to ensure the data is successfully written.

009_EncodeFromCSV

Directories

01_in_csv
02_out_edt
03_done_csv
04_err_csv

Any CGIF Flat formatted CSV file (exactly the same as produced from the output of Pipeline 004) dropped into the input directory will be translated to an EDI file and placed in the output directory. If there is an error processing the CSV file, it will be moved to the Err directory.

010_EncodeFromXML

Directories

01_in_xml
02_out_edt
03_done_xml
04_err_xml

Any CGIFXML2 Formatted XML file (the same as produced from the output of Pipeline 005) dropped into the input directory will be translated to an EDI file and placed in the output directory. If there is an error processing the XML file, it will be moved to the Err directory.

011_EncodeFromXMLDB

Directories

01_out_edt

EVENT Parameters

EVENT_DATA01 - XML_ID reference to a row within the XML_STOR table

EVENT_DATA02 - Filename of the file to create within the output directory

This is a passive pipeline in that it is not triggered via the normal background automation system, but rather a direct insert into the EVENTS table. This pipeline works similarly to Pipeline 010 in that it encodes an EDI file from XML data, but instead of sourcing it from a data file placed in a directory, it obtains it from the XML_STOR XML data storage column in the BIN_XML table.

To trigger this workflow, you will need to directly insert into the EVENTS table:

```
INSERT INTO EVENTS (EVENT_OWNER, TSRC_ID, EVENT_DATA01, EVENT_DATA02) SELECT -1, 11, 1, 'filename.edt'
```

012_EncodeFromBINDB

Directories

01_out_edt

EVENT Parameters

EVENT_DATA01 – Table name, View name or Stored Procedure to source CGIF Flat data

EVENT_DATA02 – Filename of the file to create within the output directory

This is another passive pipeline that will accept a generalized source of CGIF Flat formatted information and create an outbound EDI file.

Here are some examples of creating outbound files.

Example 1: Retrieve a file placed into the BIN system with a BIN_ID of 1, ensure proper sorted order of elements and then store it to the output directory as 'filename.edt':

```
INSERT INTO EVENTS (EVENT_OWNER, TSRC_ID, EVENT_DATA01, EVENT_DATA02) SELECT -1, 12, 'EXEC USP_FETCH_CULLED_BIN 1', 'filename.edt'
```

Example 2: Retrieve the contents of vw_Export_837P, ensure properly sorted order and store it on the output directory as 'filename.edt':

```
INSERT INTO EVENTS (EVENT_OWNER, TSRC_ID, EVENT_DATA01, EVENT_DATA02) SELECT -1, 12, 'EXEC USP_ORDEREDI ''vw_Export_837p'', 'filename.edt'
```

Although sourcing directly from a view is possible, it is recommended you use the stored procedures to properly sort the data before the Chiapas EDI engine accepts it.

013_LargeFileDecodeToBINDB_Merge

Directories

01_in_edt

02_done_edt

03_err_edt

This is a serial workflow that processes one file at a time, and will merge it into the BIN similar to the Pipeline 007 workflow. This pipeline has been optimized for very large file sizes (larger than 100 Megabytes) to minimize memory consumption and data transfers to the database. Unlike the Pipeline 007 workflow, this will completely load the BIN table by the completion of the workflow.

USER REFERENCE

This chapter will cover the essential functions of Chiapas EDI Enterprise, beginning with an in-depth tour of the different parts of Chiapas Studio and proceeding through the core functions of scheduling and automation. It will cover the Chiapas Gate Intermediate Format, which is the core naming convention and technology that Chiapas EDI uses to translate between EDI files and XML and Flat tables. The main objective of this section is to give the user an understanding of the fundamentals of EDI translation and how Chiapas EDI can be used in a variety of business scenarios involving EDI files.

What is Chiapas EDI Enterprise?

In simplest terms, Chiapas EDI Enterprise is a Windows-based ANSI X12 4010/5010 middleware EDI platform. *Middleware* refers to a category of software where the main purpose is to bridge enterprise systems with other systems. *EDI* stands for Electronic Data Interchange, which refers to the process of exchanging business information between trading partners. *Platform* means that this program allows you to create *other* programs, giving you all the resources necessary to develop and test solutions and then roll them into production.

Within an enterprise, EDI is rarely necessary because usually the different parts of the business have access to a centralized database – if all of the information is in one spot, there is no need to transform it to a different format. In the healthcare sphere, however, a ‘business’ could in fact be a complex partnership integrating dozens of different corporations. In order to fulfill the legal and contractual obligations for this partnership, it may be necessary to transmit and receive a variety of encounter data, enrollment information, healthcare claims, and remittances from a variety of sources. For these specific sets of healthcare information, Federal law mandates that the information be exchanged via the 5010 transaction sets maintained by the ASC X12 standards body and published as the HIPAA Implementation Guides.

The 5010 transactions define a dozen different types of business information exchanges, containing thousands of individual data elements. Chiapas EDI Enterprise contains sufficient information about these standards to parse them into XML and SQL table formats. The resulting formats are called *CGIF*, or *Chiapas Gate Intermediate Format*. As there are significant complexities embedded in the Implementation Guides, CGIF is designed to be capable enough of handling any valid HIPAA file, yet simple to be accessible to healthcare business analysts and developers. Once a business analyst becomes familiar with associating CGIF elements with the corresponding HIPAA elements, the process of decoding incoming EDI messages becomes much simpler.

Once the logistics of the actual decoding or encoding is worked out using the front-end *Chiapas Studio*, the developer can then use the library of included automation components to create business processes that are ‘hands-free’ – processes that run with no manual user intervention. Chiapas Studio is covered in the second chapter of the User Reference, and the automation components are covered in the third chapter.

CHIAPAS GATE INTERMEDIATE FORMAT

To understand how this technology functions, it's first necessary to review the fundamental structure of any HIPAA transaction and how these structures are used in modern healthcare communications. At the highest level of organization, all HIPAA transactions are grouped by specifications. These specifications are defined in the HIPAA Implementation Guides and sold and published by the Washington Publishing Corporation (WPC) on behalf of the ASC X12 organization. They are divided into two categories, 4010 and 5010. Each of these categories defines a specific revision of a set of healthcare communication specifications, such as claims, authorizations, enrollments and remittance advice. The 4010 set of specifications were finalized in 2002, whereas the 5010 specifications were finalized in 2008. Although 5010 deprecated the 4010 set, some trading partners have not yet converted to the new revision. Many of the transactions have one or two formally published Addenda that materially impact the specification.

Within each specification, information is grouped into hierarchical structures called loops. Loops are composed of segments, which are themselves made of elements. In certain cases, elements are further divided into composite elements. At every level, numeric and alphabetical text qualifiers are used to identify the information being described within the loops and segments. The specifications define every element possible with a given transaction, but trading partners generally need to exchange only a small fraction of these elements to meet their business requirements. Larger trading partners will often formalize the required mappings into a *Companion Guide*, which serves as an easy point of reference about what information is being exchanged.

Hierarchical protocols like XML and HIPAA are generally used to transmit groups of information in batches. However, within healthcare enterprise systems, healthcare data is almost never stored this way – often, it is embedded within relational databases where information is stored in columns and rows, on data elements specific to that enterprise. Without common implementation guides, trading partners evolve a patchwork of proprietary protocols to meet their business requirements. The purpose and role of the implementation guides is to standardize the definitions of all data elements so that all healthcare entities use a single, common language for standard transactions.

The *Chiapas Gate Intermediate Format*, or CGIF, represents the syntax that Chiapas EDI uses to identify HIPAA elements and assign human-understandable names to them. It is subdivided into two parts:

CGIF Flat – This implementation encodes HIPAA EDI elements in a flat, column / row tabular format. This is the form used for SQL tables and CSV files.

CGIFXML2 – This is the implementation used to encode HIPAA EDI elements in the XML format.

Before either of these representations can be understood, it is important to be familiar with the HIPAA Implementation Guides and how transactions are arranged by loops, segments and elements.

GENERAL CONCEPTS

Loops represent a bundle of information that is unique within a transaction. They are arranged in a hierarchical structure so that one loop may have one or more child loops. Each child loop begins with a mandatory header segment that is unique among all of the children loops belonging to that parent. Certain loops can be uniquely identified just with the 2-3 character segment code for the header segment

– for example, the CLM segment in 837 transactions only occurs for the 2300 Claim Information loop, and nowhere else in the transaction. More commonly, however, there is an element that *qualifies* the header segment and thus the loop. Although not specifically called out in the Implementation Guides, generally the first element for most loops and the third element for Hierarchical Level (HL segment code) loops uniquely define the loop. When Chiapas EDI Enterprise decodes a file, it is these qualifiers that guide it to find the right loop in which to place the encoded information.

Segments are units of business information that make up a loop. Each segment is ordered within the loop by *position* and is either unique by the segment code or by the segment code and a qualifier element located within the segment. For example, REF and DTP segments always have a qualifier element in the first element position that uniquely identify that segment within the loop.

Elements are the individual parts of the segment, sub-divided by the element separator defined by the first segment in the file. In certain situations, an element can be further sub-divided into **composite elements**, which is divided by another separator character defined within the first segment in the file.

Overloads is a concept specific to the CGIF format, and represents a way of abstracting away the housekeeping of knowing which qualifier value defines which business value. Qualifier elements are actually *metadata*, or data that describes data, and thus should be considered as part of the protocol and not having any business value by itself.

For example, let's say that within a hypothetical transaction, there exists a loop 2900 where the 2nd segment is a REF segment that can encode five different IDs, according to the qualifier in the first element:

REF Element 01	REF Element 02
AB	Shoe Size
AC	Shoe Weight
AD	Shoe Color

In an EDI file based on this fictional specification, these two segments define both a shoe size and weight:

```
REF*AB*8.5~
REF*AC*3.0~
```

In **CGIF Flat** Format, the information would be represented like so:

L2900_S02_REF02_SHOE_ID_OVL_SHOE_SIZ	L2900_S02_REF02_SHOE_ID_OVL_SHOE_WGT
8.5	3.0

In **CGIFXML2** Format, the information would be projected in this way:

```
<L2900>
  <S02>
    <REF02_SHOE_ID_OVL_SHOE_SIZ>8.5</REF02_SHOE_ID_OVL_SHOE_SIZ>
    <REF02_SHOE_ID_OVL_SHOE_WGT>3.0</REF02_SHOE_ID_OVL_SHOE_WGT>
  </S02>
</L2900>
```

CGIF Flat

The most fundamental difference between the CGIFXML2 and CGIF Flat formats are how they handle *iterations*. In CGIF Flat, this is complicated because the Implementation Guides define a number of segments and loops that can iterate many times, and these iterations need to be represented in a two dimensional format. XML, being naturally hierarchical, does not need any specific syntax to handle iterations.

X vs Y Axis

To understand how these iterations impact the CGIF Flat format, we need to understand what triggers a new row. New rows are triggered by the *deepest* loop used within the EDI file that can repeat 24 or more times. Thus, even though the 2300 Claim Information loop present within an 837 transaction can repeat many times, it is not the *deepest* loop used in the transaction – that would generally be the 2400 Service Line loop. Thus, occurrences of this loop will trigger new data rows and thus represent the ‘Y’ axis of a two-dimensional grid.

Now that we have the Y – what about the X axis? As we observed in an example in the previous section, `L2900_S02_REF02_SHOE_ID_OVL_SHOE_SIZ` and `L2900_S02_REF02_SHOE_ID_OVL_SHOE_WGT` are column names that would be used to encode data for two fictional segments. If these segments were not transmitted for that particular data row, the columns would be set to blank strings or NULL in a SQL table. When decoding an EDI file, every qualifier/identifier pair encountered is ‘bundled’ into a single column in this fashion, with an ‘OVL’ embedded in the column name itself to show a short abbreviation of the qualifier.

General Structure of CGIF Flat Fields

Here is the specific structure of any CGIF Flat field. All of these components are separated by an underscore character; brackets indicate fields that are only present depending on the situation. All of the valid CGIF Flat references are compiled into the HTML Reference guides that are present in the `c:\chiapas\docs\CGIF` directory.

Here are the components of a CGIF Flat mapping:

[<Specification Prefix>
<Loop Identifier>
[<Loop Iteration Counter or Qualifier>
S<Segment Reference>
<Segment Code & Element Identifier>
[X<Element Repeat Counter>
[<Composite Element Identifier>
[<Segment Repeat Counter>
<Element Abbreviation>
[OVL_<Qualified Overload Abbreviation>]

Specification	Transaction Set / Addenda Level
A0	4010_270
A1	4010_270_A1
B0	4010_271
B1	4010_271_A1
C0	4010_276
C1	4010_276_A1
D0	4010_277
D1	4010_277_A1
E0	4010_278_REQ
E1	4010_278_REQ_A1
F0	4010_278_RESP
F1	4010_278_RESP_A1
G0	4010_820
G1	4010_820_A1
H0	4010_834
H1	4010_834_A1
I0	4010_835
I1	4010_835_A1
J0	4010_837D
J1	4010_837D_A1
K0	4010_837I
K1	4010_837I_A1
L0	4010_837P
L1	4010_837P_A1
M0	5010_270
M1	5010_270_A1
N0	5010_271
N1	5010_271_A1
O0	5010_276
P0	5010_277
P5	5010_277CA
P7	5010_824
Q0	5010_278_REQ
R0	5010_278_RESP
S0	5010_820
S1	5010_820_A1
S5	5010_820X
T0	5010_834
T1	5010_834_A1
U0	5010_835
U1	5010_835_A1
V0	5010_837D
V1	5010_837D_A1
V2	5010_837D_A2
W0	5010_837I
W1	5010_837I_A1
W2	5010_837I_A2
X0	5010_837P
X1	5010_837P_A1

Specification Prefix

This is applicable only to the very first mapping supplied to Chiapas EDI or generated from the same. This two digit code identifies both the specific transaction being used as well as the Addenda. Here is a list of the specification prefixes:

Loop Identifier

This short sequence identifies the Loop the element belongs to. The outer envelope is identified as OEISA; the GS envelope is identified as GSHDR and the ST envelope (Transaction Set) is identified as STHDR. A close examination of the CGIF documentation would reveal some loops that are not explicitly identified within the Implementation Guides; these serve to address certain special situations that can occur in the EDI files.

Loop Iteration Counter or Qualifier

In an earlier example, we specified that the 2900 loop we defined in our example has a maximum repeat count of *five* iterations. Because this is below the threshold of 24 iterations, this loop cannot be a row trigger. However, the data still needs to be preserved in the encoding system. To handle this situation, CGIF uses an *iteration prefix* to uniquely identify elements belonging to specific loop iterations:

```
L2900_S02_REF02_SHOE_ID_OVL_SHOE_SIZ
L2900_I02_S02_REF02_SHOE_ID_OVL_SHOE_SIZ
L2900_I03_S02_REF02_SHOE_ID_OVL_SHOE_SIZ
```

Loop iteration counters like this are only used in loops where the header segment is not qualified – meaning, the initial header segment is qualified by two or more values. If it is, then the qualifier on the header segment replaces the role of the loop iteration counter:

```
L2900_XZ_REF02_SHOE_ID_OVL_SHOE_SIZ
L2900_XY_REF02_SHOE_ID_OVL_SHOE_SIZ
```

Note that some loops have single-iteration child loops. In this case, the iteration counter *carries over* to the child loops so that the child loop mappings are properly associated with the correct iteration of the parent loop.

Segment Reference

This is an S followed by a zero-prefixed two digit number that identifies the segment within the loop, with S01 being the first segment in the loop. The numbering sequence can be affected by Addenda inserting segments into the middle of the loop; in this way a small Addenda change can radically affect the mappings used for encoding and decoding. These changes are always called out in the CGIF Addenda documentation.

Segment Code & Element Identifier

This component represents the 2-3 digit Segment Code as well as the two digit, zero prefixed element position for the mapping being described.

Element Repeat Counter

The new 5010 Implementation Guide allows a select few elements to have a new feature – element repeats. In this situation, a single element or group of composite elements can iterate a number of times, with the separate iterations delineated by the Element Repeat character described within character 83 of the ISA segment. When an element repeats, the second repetition has an X02 identifier inserted after the Segment Code & Element Identifier section of the field name.

Composite Element Identifier

If the element being referenced is a composite element, a second zero-prefixed number (along with a separation underscore) follows the Element Identifier above to identify the composite element.

Segment Repeat Counter

If the segment iterates two or more times, then this component relays the iteration of that segment. Just as a loop iteration, this value is prefixed with an I followed by a zero-prefixed two digit number.

Element Abbreviation

This is an abbreviated name of the element.

Qualified Overload Abbreviation

If this mapped element is *qualified* by a value in another element (generally, the previous element acts as the qualifier to indicate what kind of identifier is present), then the mapped element will always be accompanied by a suffix that indicates the value in the qualifier element.

Examples:

Scenario 1: Claim Note Segment Iterations

The twenty-fourth segment in the 5010 837 I 2300 loop is the Claim Note segment, which can repeat up to ten times. Furthermore, Claim Notes are divided into a number of categories, including Medications, Diagnosis Description, Durable Medical Equipment / Supplies and so on. In CGIF, any information that is

qualified by a value in a preceding element is called an overload, and all overloads have their own discrete mapping. These mappings would reference the first three occurrences of a Claim Note segment containing information on Durable Medical Equipment / Supplies:

```
L2300_S24_NOT02_TEXT_OVL_DME_SUPLS  
L2300_S24_NOT02_I02_TEXT_OVL_DME_SUPLS  
L2300_S24_NOT02_I03_TEXT_OVL_DME_SUPLS
```

Scenario 2: Qualified Loops

The TPA/Broker Name (Loop 1000C) within the 4010 834 Eligibility specification is an example of a qualified loop. It contains a definition for 'Broker or Sales Office' (element 1 containing 'BO' on the header segment) and 'Third Party Administrator' (element 1 containing 'TV' on the header segment). To define both the name and the required Broker Identification Code for both of these items of information, the following four mappings should be used:

```
L1000C_BO_S01_N102_BROKR_NM  
L1000C_BO_S01_N104_TPA_BROKR_ID_OVL_FEDRL_TAXPYR_ID_NR  
L1000C_TV_S01_N102_BROKR_NM  
L1000C_TV_S01_N104_TPA_BROKR_ID_OVL_FEDRL_TAXPYR_ID_NR
```

Scenario 3: Other Subscriber Information

As stated previously, the loop iteration counter can be applied to loops that are only allowed to repeat once – this is a special situation where the iteration counter actually belongs to the parent loop. For example, if we examine Loop 2320 in 5010 837 I, 'Other Subscriber Information' in the HIPAA Implementation Guide, we would observe that it can repeat up to ten times. It has nine child loops, each of which can iterate only once each. This loop is itself a child of the 2300 Claim Information loop, so we could consider all information present within these loops to be associated with a single claim. If we have two entries of the 2320 loop and each iteration conveys an ID in the 2330C Other Payer Attending Provider, the two mappings for the 2330C State License Number associated with each iteration of the parent 2320 loop would appear like this:

```
L2330C_S02_REF02_2ND_OVL_STAT_LICNS_NR  
L2330C_I02_S02_REF02_2ND_OVL_STAT_LICNS_NR
```

Note: CGIF Flat lends itself well to a large variety of situations, with one notable exception. In certain circumstances, a large 835 file may contain many hundreds or thousands of repeating PLB elements in certain financial situations. As Chiapas EDI cannot map these on a per-row basis (they do not tie directly to a claim or service line), it must map these laterally as separate columns. As the column count could actually exceed the column limits supported by SQL Server, then CGIF Flat would not be a feasible approach for this situation and the better approach would be to work with CGIFXML2 format.

CGIFXML2 Format

This format is about encoding EDI file information into a hierarchical format known as *XML*. XML is a very well-known format in the Information Service field because it has both flexibility and the traction of many professionals having familiarity with it. Furthermore, XML format can lend itself more efficiently to a certain number of scenarios:

1. 'Free Form' EDI Data

CGIF Flat lends itself strongly to scenarios where all known data elements are planned in advance via Companion Guides shared between trading partners. When processing data in bulk, however, it may be difficult to arrive at a fixed database schema for all of the information you may need to work with. By storing XML in database tables, the information within the original EDI file can be re-queried according to different business requirements without having to reprocess the EDI files.

2. Enterprise Data Mapping

There are certain scenarios where data will exist in the enterprise system that may be difficult to translate to the special column names that CGIF Flat requires. One such example is the 'Other Subscriber Information' mentioned in the example above where there can be ten iterations of the elements within the 2320 loop and furthermore additional elements recorded in the child loops. In CGIF Flat, all of the possible mappings would be cumbersome compared to encoding this information in CGIFXML2.

CGIFXML2 mappings are quite similar to the CGIF Flat mappings except that they are 'split' to be more in line with the hierarchical structure of XML. Furthermore, although loop qualifiers are used on loops, iteration counters are not used in the CGIFXML2 naming convention. Instead, the loop or segment information simply repeats as it would in the EDI file.

The root node of a CGIFXML2 file is called 'CGIFXML2Root', and ends with an underscore and two-digit specification/addenda tag, similar to the first mapping in CGIF Flat.

Then, both loops and segment references define specific node nesting levels, so that the metadata name surrounding the actual data will comprise only of the following components:

```
<Segment Code & Element Identifier>
[X<Element Repeat Counter>]
[<Composite Element Identifier>]
<Element Abbreviation>
[OVL_<Qualified Overload Abbreviation>]
```

Here is an example of a CGIFXML2 file that defines eligibility for a single person. This is not a valid file, as most fields have been omitted in order to emphasize the structure:

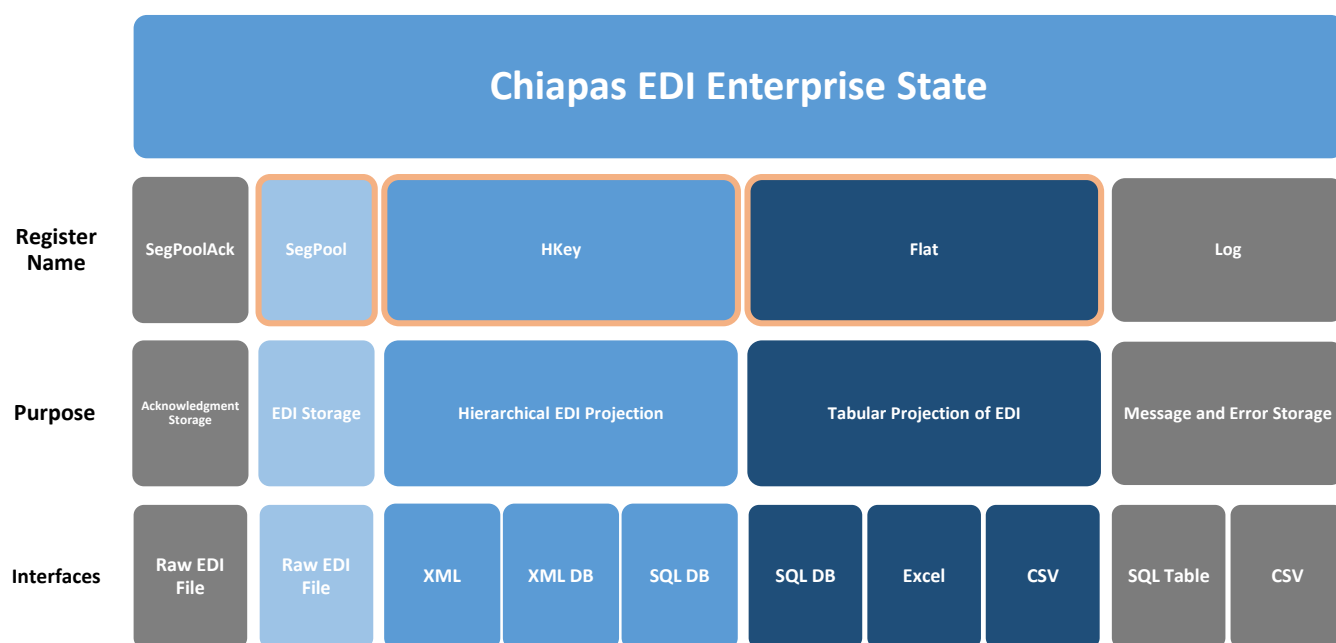
```
<CGIFXML2Root_T1>
  <OEISA>
    <S01>
      <ISA02_AUTH_NFO_OVL_NO_AUTH_NFO>.....</ISA02_AUTH_NFO_OVL_NO_AUTH_NFO>
    </S01>
    <GSHDR>
      <S01>
        <GS02_APP_SENDR_CD>SENDR_ID</GS02_APP_SENDR_CD>
      </S01>
      <STHDR>
        <S01>
          <ST02_CONTRL_NR>100000000</ST02_CONTRL_NR>
          <ST03_IMPLMNTN_CONVNTN_REFNC>005010X220A1</ST03_IMPLMNTN_CONVNTN_REFNC>
        </S01>
        <S02>
          <BGN01_TS_PURPS_CD>00</BGN01_TS_PURPS_CD>
        </S02>
        <L1000A>
          <S01>
            <N102_PLAN_SPONSOR_NM>CHIAPAS EDI TECH</N102_PLAN_SPONSOR_NM>
          </S01>
        </L1000A>
        <L1000B>
          <S01>
            <N102_INSR_NM>DOOLITTLE HOSPITAL</N102_INSR_NM>
          </S01>
        </L1000B>
        <L2000>
          <S01>
            <INS01_MEM_IND>Y</INS01_MEM_IND>
          </S01>
          <S02>
            <REF02_SUB_ID_OVL_SUB_NR>987654320</REF02_SUB_ID_OVL_SUB_NR>
          </S02>
          <S03>
            <REF02_MEM_GRP_POLICY_NR_OVL_GRP_POLICY_NR>
            BASICHEALTHCAREPLAN</REF02_MEM_GRP_POLICY_NR_OVL_GRP_POLICY_NR>
          </S03>
          <S04>
            <REF02_SUPP_ID_OVL_CLIENT_REPRPNG_CATGRY>
            100001</REF02_SUPP_ID_OVL_CLIENT_REPRPNG_CATGRY>
          </S04>
          <S05>
            <DTP03_STATS_NFO_EFF_DT_OVL_ELIG_BEGN_D8>
            20120101</DTP03_STATS_NFO_EFF_DT_OVL_ELIG_BEGN_D8>
          </S05>
          <L2100A_IL>
            <S01>
              <NM103_MEM_LNAME_OVL_PERSN>BURGER</NM103_MEM_LNAME_OVL_PERSN>
            </S01>
            <S02>
              <PER04_COMM_NR_OVL_HOM_PHON_NR>
              4155551200</PER04_COMM_NR_OVL_HOM_PHON_NR>
            </S02>
            <S03>
              <N301_ADRS_LIN>123 A ST</N301_ADRS_LIN>
            </S03>
            <S04>
              <N401_CITY_NM>SAN FRANCISCO</N401_CITY_NM>
            </S04>
            <S05>
              <DMG02_BIRTH_DT_OVL_DT_CCYYMMDD>
              19800707</DMG02_BIRTH_DT_OVL_DT_CCYYMMDD>
            </S05>
          </L2100A_IL>
        </L2000>
      </S01>
      <SE02_TCN>100000000</SE02_TCN>
    </S01>
  </STHDR>
  <S03>
    <GE02_GCN>1</GE02_GCN>
  </S03>
</GSHDR>
<S03>
  <IEA02_ICN>100000000</IEA02_ICN>
</S03>
</OEISA>
</CGIFXML2Root_T1>
```

From this example, we can observe how the data is arranged. Note that L2100A, being a qualified loop, possesses the 'IL' qualifier suffix in this example.

CHIAPAS EDI ENTERPRISE Registers and Runtime Environment

Both the Chiapas Studio and the Worker Processes that make up the automation environment share something in common: a runtime *state* object. The state is a collection of various registers, variables and database connections that make up the Chiapas EDI runtime environment. The state is transparent to the end user – it can only be accessed indirectly by changing the registers, opening and closing connections and so on.

Here are the registers as well as the primary interfaces they connect to:



Emphasis is added to the SegPool, HKey and Flat registers to highlight the conversion path between them. These three registers are said to be *equivalent* – meaning, they can translate freely data between them without loss of data. If an incoming EDI file has integrity errors, however, this can cause Chiapas EDI Enterprise to omit some elements or segments as it tries to translate the file, although these events will always be flagged in the notification log.

Chiapas EDI Enterprise is purposefully lenient when decoding incoming files – as long as it has sufficient information to parse the basic structure of the file, it will keep on parsing even as it flags any syntax violations it encounters. However, Chiapas EDI Enterprise does not execute any transformations on the data – all internal information is in ‘string’ format that appears exactly as it does within the EDI file, and it is up to the user to translate dates into the 8 digit format suitable for EDI, such as ‘20140625’.

SegPool

This register is a simple representation of a stream of segments. Depending on the direction of the workflow, it serves to receive data before saving to a new EDI file, or accepting the contents of an existing file for further processing later on. It can be directly translated to and from the HKey register.

HKey

This register is Chiapas EDI Enterprise's internal hierarchical storage mechanism for EDI information, and it can accept data or send data in both directions with the SegPool and Flat registers. This translation process is completely *black boxed* – nothing about it can be managed directly by the user. The only way to influence the behavior of the translation process is by managing the data fed into it.

The HKey register has three bi-directional interfaces to end user systems:

XML - CGIFXML2 formatted XML files stored on the filesystem

XML DB – CGIFXML2 formatted XML stored in database tables

SQL DB – The HKey register can also interact with database tables directly, with a database table representing a single loop and all tables are tied together with local and foreign keys.

Flat

The Flat register is usually the easiest way business customers can interact with EDI. It is the least efficient in terms of memory storage, as every row must contain every mapping for the whole EDI file, but also often the easiest to create and maintain for a variety of scenarios. Each mapped element is represented via a fixed, non-user-definable column name that describes the loop, segment and element positioning. These column names are quite verbose and can be over sixty characters in length. The first mapped element name also contains a special prefix that defines the specification associated with the mappings.

Log

Chiapas EDI Enterprise keeps an active record of all errors and user-generated messages in the Log Register. Specifically, if an error occurs during decoding or encoding, it will be recorded here.

SegPoolAck

This is a representation of a 999 Acknowledgment message – since it works in conjunction with the SegPool register, it needs a separate space. Chiapas EDI Enterprise has a basic capability of generating 999 messages: if a file is received and decoded without critical error, it will generate a 999 message accepting every transaction that was transmitted. If decoding stopped on a critical error, Chiapas EDI Enterprise will generate a 999 message indicating where the error was encountered. This 999 can then be returned to the trading partner.

The purpose of a 999 message is generally as an automatic mechanism for showing the transaction was accepted by the trading partner.

Chiapas Studio also has a capability of parsing an incoming 999 against an EDI file so that the 999 Messages are attached to specific lines in the EDI file.

CHIAPAS STUDIO INTERFACE

Chiapas Studio serves three purposes:

- Function as a development and testing area to work with the translation engine and see immediate feedback
- Develop and debug automation workflows
- Create new automation triggers and see the results of previous workflow executions

These three areas are accessed between three separate interfaces accessed via the pulldown 'Window' menu at the top of the screen.

The main Studio is a workspace for directly accessing the engine interface features and to test your transformations. Some features of Chiapas Studio:

- Contextual error highlighting on the SegPool window
- Provided the registers do not exceed certain size thresholds, Chiapas Studio will display the data contained within each of the three registers.
- All register I/O interfaces are directly accessible

For a visual identification of the different areas of the Studio interface, see the beginning of the Quick Start Guide. This overview will go into more depth and enumerate the full functionality of the Studio interface.

Register Information

Flat	0000x000000	Encode
HKey	000	Decode
Error Count	0000	
SegPool	000000	HIG Page: ---

Flat – The current status of the Flat register. The four digit number displays the number of columns, with the second number displaying the number of rows.

HKey – The number of loops stored in the HKey register.

Error Count – The number of messages stored in the Log register

SegPool – The number of segments stored in the SegPool register

HIG Page – When working in the Flat Register pane, clicking on individual mappings will also set the HIG Page value here. This is the page reference in the related original HIPAA Implementation Guide where that element is defined.

Encode – This translates a loaded HKey register to the SegPool register.

Decode – This translates a loaded SegPool register to the HKey register.

Data I/O Pane

This area focuses on providing import and export capabilities to registers within the Chiapas Studio state. The top row of tabs identify the register, whereas the sub-tabs identify individual I/O interfaces relating to that register.

Flat Register Interfaces

DB Table Interface

DB Table | CSV | Excel

☐ Flat To Table Execute

☒ Table to Flat ForceMerge Merge

DB Type: SQL Server

Data Source: Data Source=.;Database=chiapasData;Trusted_Connection=true;

Table: vw_Export_834

This pane enables the user to import data from or export data to a database. Interfaces for ODBC, Ole, SQL Server and SQLite are presented. All data pulled externally is expected to be in CGIF Flat format; similarly, all data exported will be the same.

Flat to Table – Radio button option for exporting Flat register to database

Table to Flat – Radio button option for importing the Flat register from a database

DB Type – Sets the database type.

Data Source – The Data Source field must be populated with a valid connection string, as it is known in the .NET development space. An excellent resource on this topic is the website at: <http://connectionstrings.com>

The default Data Source string assumes that there is a local server with a default name containing the distribution chiapasData database.

Table – For operations storing data to Table, this must be a table name.

For operations loading data from a table, then the Table field can be filled with one of the following:

- A table name
- SELECT statement
- EXEC stored procedure statement that returns a dataset

Note that SELECT should be followed by ORDER BY clauses in order to guarantee proper ordering of the incoming data. Also, if a SELECT or EXEC clause is used, it is case sensitive so they should be all CAPS.

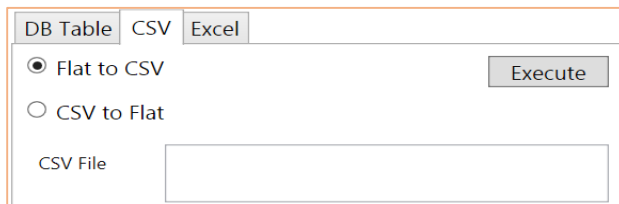
Execute – This carries out a simple store or load operation, as indicated by the radio buttons. When the operation is set to save data to database to SQLite and SQL Server databases, the Execute option will create the destination table if it doesn't exist. Then, the data is then shuttled to the destination via a series of INSERT statements. If there are fields in the register that don't exist in an existing destination table, the operation will fail.

As shown above in the Table description, when data is being loaded to the Flat register, a variety of SQL Server data sources is supported.

Merge – This is a store-only button that only functions on SQLite and SQL Server destinations. If there are fields present on the register not present in the destination table, then these fields will be logged as 'Unmerged Mapping' error messages to the Log and these fields will be omitted from the INSERT statements, allowing the operation to succeed.

ForceMerge – This is a store-only button that only functions for SQL Server destinations. If there are fields present on the register not present in the destination table, then Chiapas EDI Enterprise will execute a series of SQL ALTER TABLE commands to add the missing columns to the destination table prior to inserting the data, allowing the operation to succeed without omitting any information.

CSV Interface



Flat to CSV – Radio button to select exporting the Flat register to CSV format

CSV to Flat – Radio button to select importing data from a CSV file to the Flat register

CSV File – Text box for the full path and filename of the CSV file. Double clicking on the window will bring up an Open File or Save File dialog, depending on the radio button set.

Execute – Carries out the operation. Note that CGIF Flat formatted CSV files obey the following rules:

- First line contains column header
- All elements (including column headers) are surrounded by double quotes and comma separated

Excel Interface



Flat to Excel – This will save the Flat register to an Excel spreadsheet.

Flat from Excel – This will load the Flat register from the first worksheet of an Excel spreadsheet.

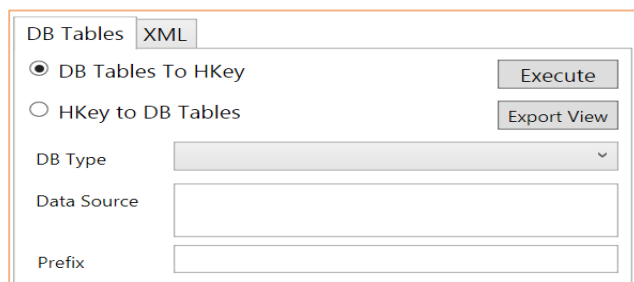
Excel File – This is the filename of the Excel spreadsheet. Double clicking in the window will bring up an Open File or Save File dialog.

Execute – This will carry out the specific operation. For save operations, the extension of the filename will help the engine to determine the file type: .XLS files will be saved as the old Excel '97 format, whereas .XLSX files will be saved with the newer Excel 2003 format. The Excel 97 format has hard limitations on the number of rows and columns that can be stored and is not recommended.

HKey Register Interfaces

The HKey Data I/O tab enables easy loading and saving of the HKey register. It supports saving to DB Tables and XML; the XML DB interface is supported only within Automation.

DB Tables

The screenshot shows a software interface with two tabs: 'DB Tables' and 'XML'. The 'DB Tables' tab is active. It contains two radio buttons: 'DB Tables To HKey' (selected) and 'HKey to DB Tables'. To the right of these are two buttons: 'Execute' and 'Export View'. Below the radio buttons are three input fields: 'DB Type' (a dropdown menu), 'Data Source' (a text box), and 'Prefix' (a text box).

This tab represents a transport mechanism between SQL database tables and the HKey register. When exporting a loaded HKey, the engine will create a number of database tables with the indicated prefix and then populate them with data. The data will be formatted in a way very similar to CGIF Flat, except that all loops are isolated from one another and furthermore, each table will have a primary key called KEY_ID (type of INT) and a foreign key called PAR_KEY_ID (type of INT). All other fields are typed VARCHAR(200).

DB Tables to HKey – Radio button to select loading the HKey from DB tables

HKey to DB Tables – Radio button to select saving the HKey to DB tables

DB Type – Database type

Data Source – Data Source string

Prefix – These characters act as a prefix to the loop names.

Export View – This button will create a SQL Server database view called 'vw_HKey_<prefix>' that will link all of these tables together into a CGIF Flat-compatible format.

Execute –

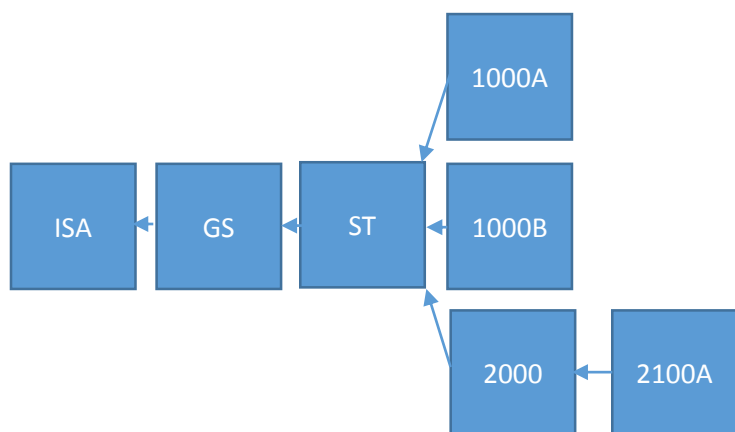
For loading the HKey from DB Tables, the engine will first load in the <prefix> + OEISA table and use the first column specification prefix tag to derive the specification. It will then attempt to load all loops defined for that specification from the database tables. Assuming the data is properly formatted, the HKey will be loaded. Although this operation is far more complex than loading a CGIF Flat to the Flat register and then transforming it to the HKey, directly loading the HKey from tables like this also means there are time and performance advantages as CGIF Flat has a large amount of duplicated data.

For saving the HKey to DB Tables, the engine will automatically create all of the tables (properly prefixed as specified) and populate them with data.

The KEY_ID is a unique row identifier for each row; the PAR_KEY_ID field is a reference to the KEY_ID of the parent loop-table. When the PAR_KEY_ID repeats for multiple rows, this means that there are multiple iterations of that loop for a particular parent loop, such as service lines to claims.

Example:

Supposing we had a simple 834 file, and we decoded it to an HKey and then stored it as database tables with a prefix of 'X'. The tables would be called XOEISA, XGSHDR, XSTHDR, XL1000A, XL1000B, XL2000, and XL2100A. Each row in each table would have been uniquely identified with the KEY_ID field. The PAR_KEY_ID would point to the KEY_ID field of the *parent* table, as shown below:



XML

DB Tables XML

☒ XML To HKey ☐ HKey to XML

Execute

XML File

This interface provides XML transport for the HKey register. For an example of the CGIFXML2 format Chiapas EDI Enterprise uses, see the chapter on the Chiapas Gate Intermediate Format.

XML to HKey – Radio button to select loading the HKey from XML

HKey to XML – Radio button to select saving the HKey to XML

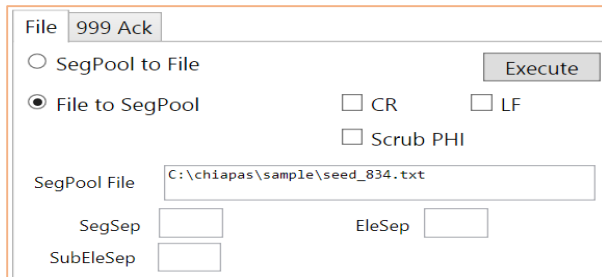
XML File – Textbox to indicate the XML filename. Double clicking in the window will bring up an Open File or Save File dialog.

Execute – This will carry out the indicated operation.

SegPool Register Interfaces

This tab presents an interface for loading HIPAA transaction EDI files from and to the SegPool register.

File



This interface allows you to adjust the formatting of the file, including the Segment Separator, Element Separator, Composite Element Separator, and whether it has Carriage Return / Line Feed formatting at the end of each segment. Also, it gives an option to scrub Protected Health Information from the file prior to saving.

SegPool to File – Exports the loaded SegPool register

File to SegPool – Imports an EDI file to the SegPool register

CR - Setting CR will append a Carriage Return (ASCII 13) character to each segment.

LF - Setting LF will append a LineFeed (ASCII 10) character to each segment.

Scrub PHI - Setting this option before exporting will scrub it of Protected Healthcare Information. Dates, Names and Identifiers will be altered from the originals. Note that this is merely a ‘best guess’ to filter out the most common pieces of publically identifiable information; the only way to guarantee a file contains no PHI is to manually review and alter the file.

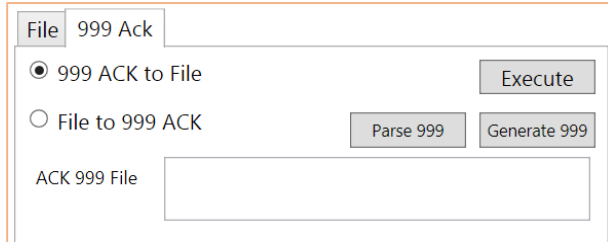
SegPool File – File name of the file to be loaded or saved. Double clicking on the textbox will bring up an Open File or Save File dialog.

SegSeg – Segment Separator.

EleSep – Element Separator.

SubEleSep – Sub-Element Separator

999 Ack



The screenshot shows a software interface for handling 999 transactions. It features a 'File' button at the top left. Below it, the text '999 Ack' is displayed. There are two radio buttons: the first is selected and labeled '999 ACK to File', and the second is labeled 'File to 999 ACK'. To the right of the first radio button is an 'Execute' button. Below the second radio button are two buttons: 'Parse 999' and 'Generate 999'. At the bottom, there is a label 'ACK 999 File' followed by a large, empty rectangular text input field.

Chiapas EDI 2012 has limited built-in support for 999 transactions. A 999 transaction is generally a tool clearinghouses and trading partners use to generate automated notifications about the status of an incoming file. Generally, 999's indicate whether the file has been rejected at an EDI syntax level or that the file has been successfully parsed and is accepted for further processing. The 999 does not communicate information about the status of a claim or any specific business context.

Chiapas EDI 2012 can generate a 999 after a Decode event: either a simple 999 'Acceptance' record will be created if the file parsed without major errors, or it will be Rejected for a serious integrity violation. For incoming 999 messages, Chiapas EDI 2012 can apply the 999 against an existing loaded SegPool and generate messages that can be displayed alongside the segments in the SegPool display, allowing you to see which segments triggered the errors that were listed within the 999.

Parse 999 – This will load a 999 file and add messages to the active Message Log that will be displayed in the SegPool pane.

Generate 999 – This will generate a basic 999 acknowledgment file indicating 'Accepted' when a Decode event is able to parse a file, or Rejected in the case of a serious integrity error.

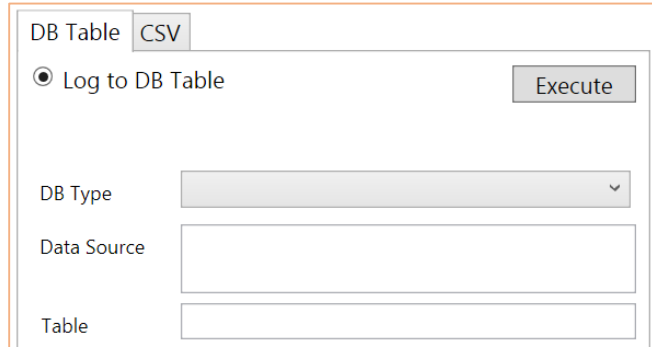
Execute – This will save a 999 file that was previously generated via the Generate 999 function, or it will load in 999 Transaction Acknowledgment file that was generated from a clearinghouse.

ACK 999 File – The file to be saved or loaded. Double clicking here will bring up an Open File or Save File dialog.

Log Register Interface

Chiapas EDI Enterprise does not support loading a Log into the register; its purpose is to be a write-only record of events that occur during the translations. In Chiapas Studio 2014, you have the option of writing the log out to a file at any time. This register tracks all messages and errors that can occur during encode or decode processes, including critical integrity errors or more minor syntax errors.

DB Table



DB Table CSV

☒ Log to DB Table Execute

DB Type

Data Source

Table

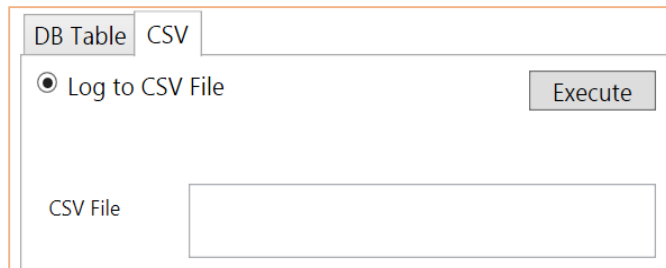
DB Type – Database type to receive the Log register.

Data Source - Connection string to the receiving database.

Table – Name of the table to receive the Log register.

Execute – Exports the Log register to a database table. The database format follows that of the MSGLOG table included in the chiapasData database that is part of the distribution, which is covered in the Technical Reference. If the table does not exist, it will be created.

CSV



DB Table CSV

☒ Log to CSV File Execute

CSV File

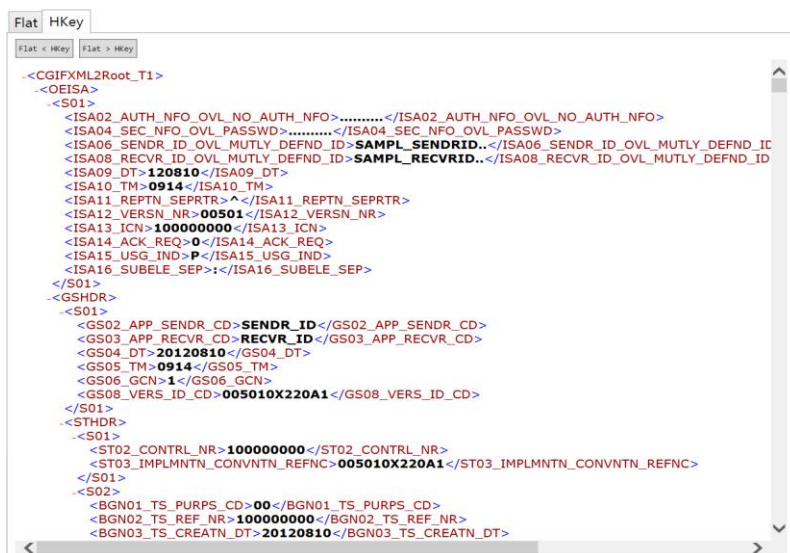
CSV File – CSV file to create when exporting the register.

Execute – Exports the Log register to a CSV file. It also will follow the field naming convention of the MSGLOG table.

Flat/HKey Register Pane

The central two windows of Chiapas Studio are dedicated to providing a live view of the information contained within the Flat and HKey registers. Because these displays are memory and processor intensive, they will not work to display data files larger than 10 MB. Even if the displays do not function, the engine is still generally able to carry out functions on the registers normally.

HKey Register



This tab displays information about a loaded HKey register. As described in a previous section, the HKey stores hierarchical business information and is an intermediary between the SegPool register (for interacting with HIPAA files) and the Flat register (for interacting with databases). The information is displayed on-screen in the form of an XML file. There are two controls that allow for translating the register to or from the Flat register.

Flat < HKey – This button translates the HKey to the Flat register.

Flat > HKey – This button loads the HKey from the Flat register.

Flat Register

This interface enables the user to view the Flat register interactively as well as modify both the mappings and the information contained within. This capability allows the user to test different situations with data and mappings without refreshing from a data source. Note that generally, the Flat is loaded either from the Data I/O interface or from the HKey pane, and then this interface is updated with a snapshot of the data. Changing the data and mappings does not directly affect the Flat register; the > Flat and Set buttons are used to actually update the internal register.

Set - This button propagates the mapping indicated in the Mapping Interface to the currently selected column within the Flat interface.

+Row - This will clone the current row and insert a duplicate.

-Row - This will remove the currently selected row.

+Col - This will insert a new, 'Unmapped' column. You must manually 'Set' it to a valid mapping before this can be used with the Chiapas Gate engine for encoding.

-Col - Removes the currently selected column.

Display - This toggles the interface to display the entire mapping name in each column, or to keep them fixed length and display only the first 15 character or so of each column.

>Flat - This button updates the Flat register from the interface.

HIG - If the HIPAA Implementation Guides have been properly configured with Chiapas Studio, then this will launch a PDF Reader application and open the HIG to the correct page for the currently selected mapping within the Flat interface.

Mapping Interface

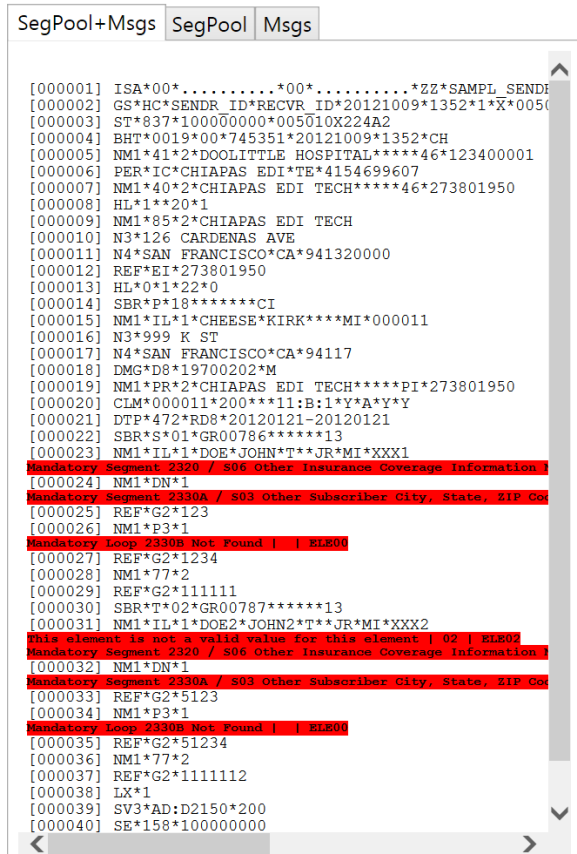
CHIAPAS
EDI TECHNOLOGIES, INC.
Healthcare Integration Solutions ©

Specification: 5010_834_A1
5010 Benefit Enrollment and Maintenance

Loop	2100A:MEMBER NAME	Ovl	IL:Insured or Subscriber
Segment	04:N4:Member City, State, ZIP Code	Iteration	
Element	02:Member State Code	Sub Element	
Overload			
CGIF Reference	L2100A_IL_S04_N402_STAT_CD		

The mapping interface has two functions. For existing mappings selected in the Flat pane, it displays verbose information about all the components of that mapping. Otherwise, it presents an interface for discovering mappings. When entering new mappings, it's important to follow this sequence: Loop, Segment, Element, and Sub-Element. The selection of one combo box dynamically changes the contents in the next combo box in sequence, so it is important to select the boxes in order from top to bottom. This interface is meant for reference only; the authoritative source for CGIF Reference mappings are the HTML files residing in the C:\chiapas\docs\CGIF directory.

SegPool Register Interface



The SegPool Display is responsible for displaying both the segments within the SegPool object as well as the Log Register. Normally, messages are associated with specific segments, so the default mode is to display both simultaneously, as shown here. However, the SegPool and Messages can each be viewed in isolation by switching to other tabs.

Menu Options

This section describes the menu bar options at the top of the window.

Session

Load Session – Loads a previously saved session

Save Session - This feature lets you save the settings within the Data I/O panel to a file that can later be restored. There is also a 'default.winstat' file within the Chiapas/bin directory that represents the default state of these settings; it can be overwritten so that every time Chiapas Studio is launched, these settings are loaded at the beginning of the session.

Copy CGIF to Clipboard - Copies the currently selected CGIF mapping to the clipboard.

Open HIG - If there is a currently selected mapping and the options have been set properly to link to a set of HIPAA Implementation Guide PDFs (available from Washington Publishing Company, <http://wpc-edi.com>), this will open the PDF to the page number describing the mapped element.

Quit - This quits out of the application.

Data I/O, Flat, HKey

These menu options provide an alternate path to access commonly used Data I/O functions. In addition, it adds one new function:

Flat Maps > Clipboard - If there is a loaded Flat register, this will create a four column description of the mappings contained within that register and paste it to the clipboard. The Seed specifications at the end of this manual were created with the help of this function.

Window

Studio Interface

Workflow Interface

Scheduler Interface

These menu options are available throughout the Chiapas EDI Enterprise, and allows you to freely switch between these interfaces. Note that files and work is not saved when you leave an interface.

WinSCP Sessions – This launches the WinSCP client that comes bundled with Chiapas EDI Enterprise. The ChiapasGate Workflow activities as well as the Scheduler system use this bundled client for remote SecureFTP operations. Note that when configuring WinSCP session, the password needs to be saved.

Help

Help System – This will bring up the online documentation based on the active window.

Set HIG Bindings - Chiapas Studio can link directly to the HIPAA Implementation Guides to pull up a page associated with any mapped element. These guides are copyrighted, controlled materials and not included with the Chiapas EDI Enterprise distribution; they must be purchased directly from Washington Publishing Corporation (wpc-edi.com). This feature works only with the initially published 4010 and 5010 PDFs that were not corrected with the Addenda. Type in the path to the file with each associated specification. In the picture below, each PDF is present along with the original PDF filename for that specification.

Window HIG Settings

Spec	Filename
4010_270	C:\chiapas\hig\X092.pdf
4010_271	C:\chiapas\hig\X092.pdf
4010_276	C:\chiapas\hig\X093.pdf
4010_277	C:\chiapas\hig\X093.pdf
4010_278_REQ	C:\chiapas\hig\X094.pdf
4010_278_RESP	C:\chiapas\hig\X094.pdf
4010_820	C:\chiapas\hig\X061.pdf
4010_834	C:\chiapas\hig\X095.pdf
4010_835	C:\chiapas\hig\X091.pdf
4010_837D	C:\chiapas\hig\X097.pdf
4010_837I	C:\chiapas\hig\X096.pdf
4010_837P	C:\chiapas\hig\X098.pdf
5010_270	C:\chiapas\hig\x279.pdf
5010_271	C:\chiapas\hig\x279.pdf
5010_276	C:\chiapas\hig\x212.pdf
5010_277	C:\chiapas\hig\x212.pdf
5010_278_REQ	C:\chiapas\hig\x217.pdf
5010_278_RESP	C:\chiapas\hig\x217.pdf
5010_820	C:\chiapas\hig\x218.pdf
5010_834	C:\chiapas\hig\x220.pdf
5010_835	C:\chiapas\hig\x221.pdf
5010_837D	C:\chiapas\hig\x224.pdf
5010_837I	C:\chiapas\hig\x223.pdf
5010_837P	C:\chiapas\hig\x222.pdf

GUI Options –

1. Refresh Segment Pane on Encode

This checkbox controls whether the Segment Pane is displayed after an Encode operation. If you are working with extremely large files, this can prevent long freezes within the user interface.

2. Refresh HKey Pane on Decode

Whenever a HIPAA file is decoded, the XML within the HKey pane is automatically refreshed. If you are working with HIPAA files more than 5 MB in size, this can lead to user interface freezes. Deactivating the interface has no bearing on the underlying registers working properly.

3. Enable Flat Data Pane

Again, when working with large data sets, disabling the display refresh can increase user interface performance.

Clear Notification Area – This will reset the notification window in the upper right hand corner.

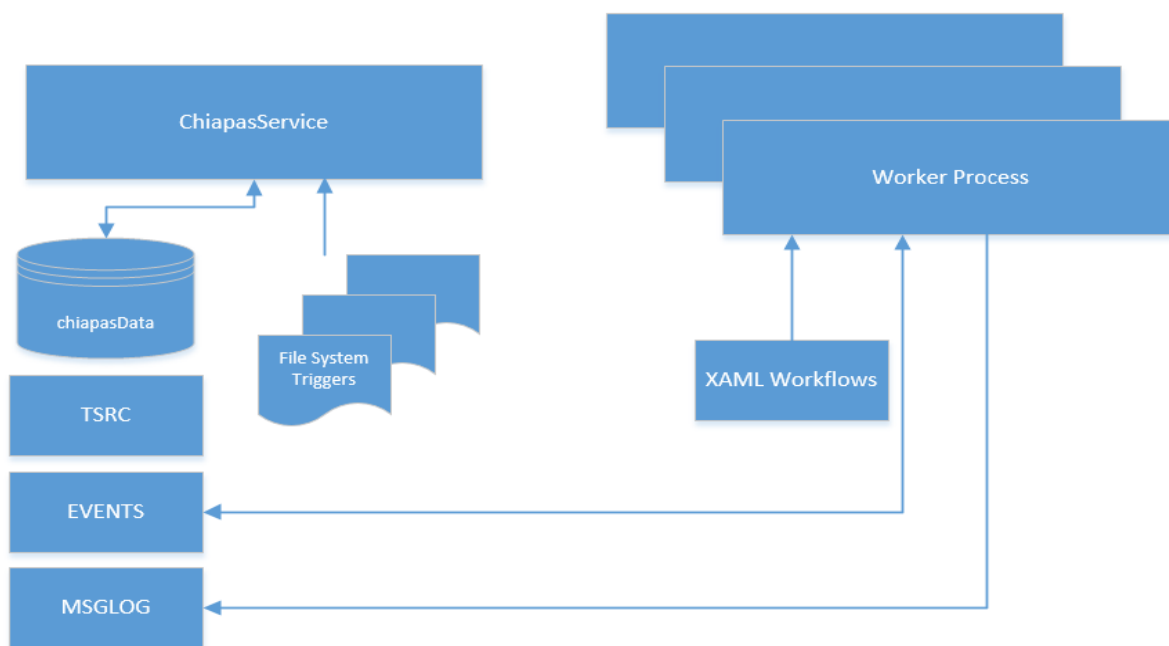
Clear Messages – Clear the message log register.

Reset All – This will completely reset the internal state object. When handling different files, it is normal to select this option to make sure no error messages or registers are left over from a previous operation.

WORKFLOW INTERFACE

Chiapas Studio is useful to handle ad-hoc tasks as well as test encoding and decoding processes. For automated processes, Chiapas EDI has an automation platform built upon a Microsoft technology called Windows Workflow Foundation 4.0. Chiapas EDI Enterprise has a basic implementation of this platform that focuses on short, mainly sequential workflows with enough logic to handle error handling and messaging. Each workflow is composed of a number of Activities, which are basic units of functionality. Chiapas EDI Enterprise supplies 41 proprietary Activities in addition to 15 basic flow-control activities. These Chiapas Activities cover all of the functionality present within the package, including support for external shell commands, database interaction, SecureFTP and File transfers.

Automation Architecture



To understand the function of the workflows, it is first necessary to understand the Automation system. Every installation of Chiapas EDI Enterprise sets up a background service called 'ChiapasService'. When the service is first started, it launches a number of background Worker Processes (depending on licensing tier). Then, the service continuously scans the TSRC table for trigger conditions. The two main trigger conditions are UPLOAD directory scans and ARCHIVE directory scans, where new files that appear in certain directories meet the criteria for 'firing' the trigger.

When the trigger is fired, the service inserts a new entry into the EVENTS table with all the information needed to start the event. This includes the filename of the file that launched the trigger, the ID of the trigger and a -1 in the EVENT_OWNER field. In the meantime, the background Worker Processes are continuously scanning the EVENTS table for new events with an EVENT_OWNER of -1. When the process finds such an EVENT, it attempts to take ownership of the event. When the event is successfully owned by the process, it goes on to handle it.

All triggers have an associated workflow, an XAML workflow file stored on the filesystem (the distribution automation workflows are placed in C:\chiapas\xaml). The Worker Process loads the XAML file and then executes it while passing arguments to the workflow that are specific to that event.

When the workflow is complete, the Worker Process will automatically write all log messages to the MSGLOG table and also update the status of the event to one of the following states:

COMPLETE, NO LOGS – The event completed normally but did not generate any messages.

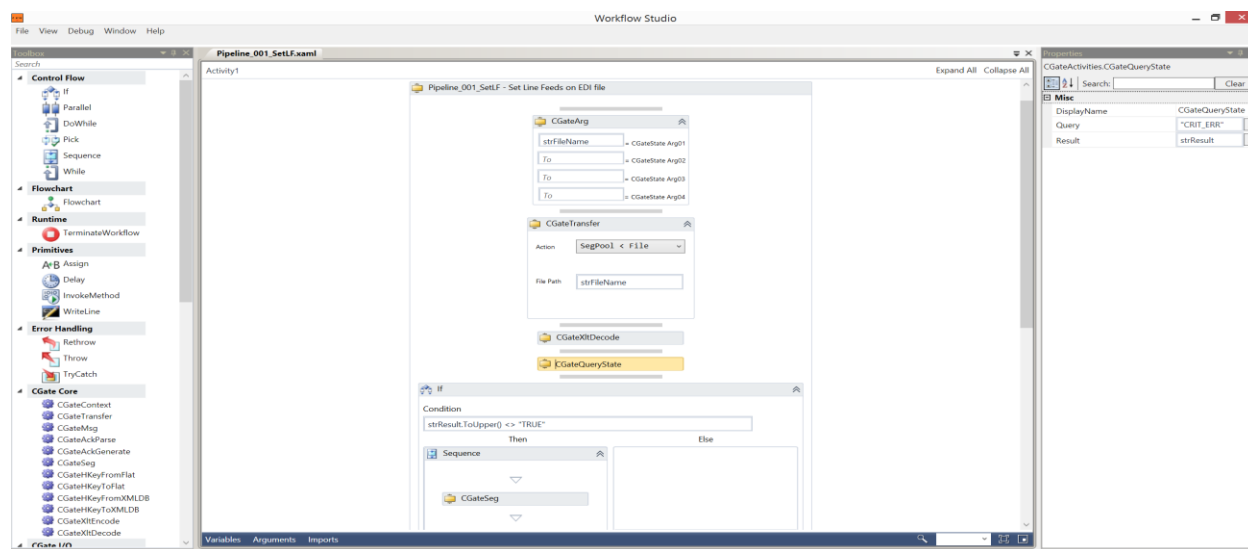
COMPLETE WITH MESSAGE LOGS – The event completed and logged some messages.

CRITICAL ERROR, SEE MESSAGE LOGS – There was a critical error during execution which was logged as a message.

THREAD ABORT ERROR - SEE C:\CHIAPAS\ERRLOG FILE – There are certain situations that will stop the normal execution of the workflow. These include low memory conditions on the server, or if the workflow spawns an external shell process that crashes and does not execute normally. The cause of the error will be written to disk to the C:\chiapas as a time-stamped ERRLOG text file.

Note: Workflows handle a single file at a time. A single launch of the Worker Process should not decode or encode files within a loop as the internal state object was designed to handle only a single encode and decode operation at a time. However, with many Worker Processes running simultaneously, it is quite possible to process large volumes of files in a short time, depending on the speed of the server and the number of available cores.

Using the Workflow Studio



The Workflow Studio is a re-hosted Workflow Designer based on Windows Workflow Foundation 4.0, and is designed to support the creation, testing and deployment of Chiapas EDI Enterprise workflows. The left pane is the *toolbox* and contains all of the Activities that are supported in this environment. They are added to the main window area via a drag-and-drop interface – the small gray bars we see between the activities in the above screenshot are areas where new activities can be inserted.

This middle pane is the *workspace* and the main work area for developing workflows. All Chiapas EDI Enterprise workflows are contained within a single container called a CGateContext Activity. This is a

custom *sequence* which can hold a stack of Activities; they will be executed in order from top to bottom. The Context activity contains the Chiapas EDI Enterprise state object; the CGateContext activity cannot be nested in other contexts and must be the root activity in the workflow.

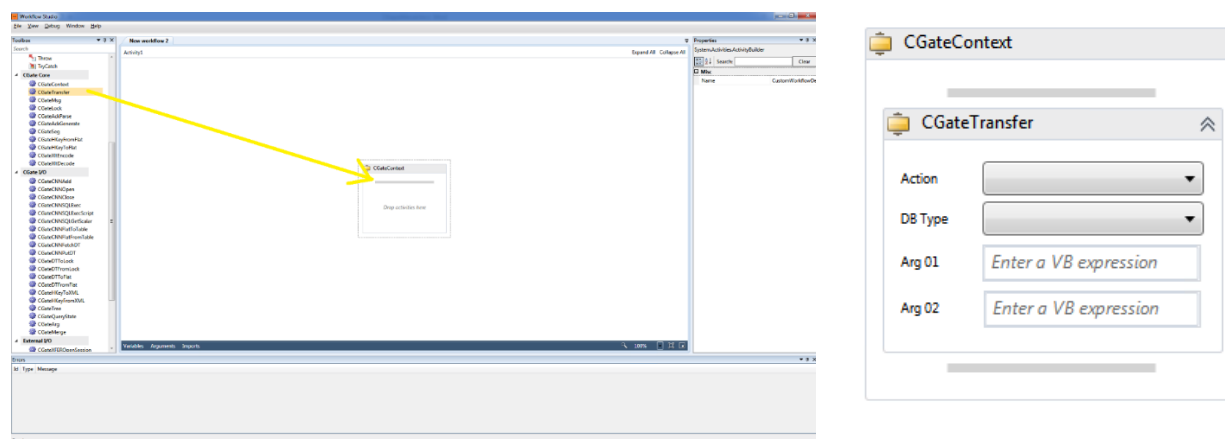
The right pane displays any specific information associated with a selected activity. For Chiapas EDI Enterprise activities that do not present a specific interface, any specific variables or settings will be accessible here.

Workflow Tutorial

To really explore how to use the Workflow Studio, let's construct a simple workflow step-by-step from scratch. In this example, we will load in one of the sample file HIPAA files, decode it, and export the results to a CSV file in the C:\chiapas directory.

Open the Workflow Studio. When you first launch Chiapas Studio, select 'Window' from the top menu options and select, 'Workflow Interface'.

Create a new workspace. Select 'File' from the menu options, and then select 'New Workflow...'



Click on the CGateTransfer activity listed under the CGateCore category in the toolbox on the left, then hold and drag the cursor to blank area in the CGateContext container in the center of the screen. Release the mouse button. This will result in adding the CGateTransfer activity to the workflow, as shown above and to the right.

Now, set the options on the CGateTransfer activity. Some activities have *designers*, which are dynamic interfaces with visual options to set on the activity itself. In this case when the `SegPool < File` action is selected, unused options will disappear.

Action: `SegPool < File`

File Path: `"C:\chiapas\sample\seed_834.txt"`

Next, add the CGateXltDecode activity by dragging the icon from the left toolbox to the gray bar underneath the CGateTransfer activity. No options need to be set for this activity.

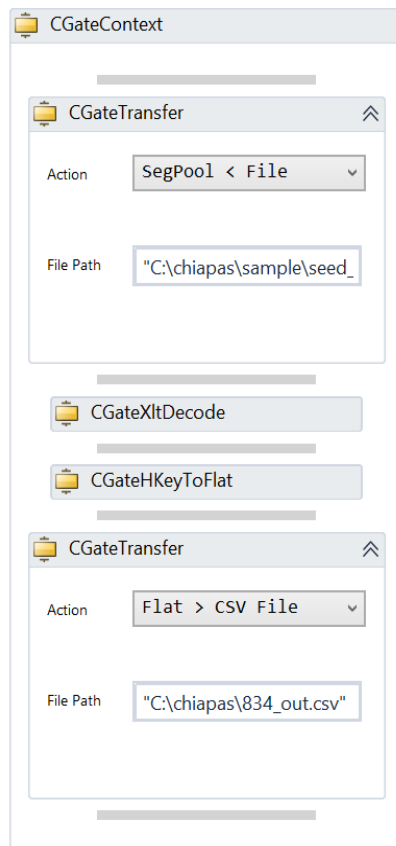
Next, add the CGateHKeyToFlat activity.

Finally, add another CGateTransfer activity with the following options:

Action: Flat > CSV File

File Path: "C:\chiapas\834_out.csv"

The result will appear like this:



Now, run the workflow by going to the Debug menu at the top of the screen and select 'Start Debugging'. After a moment, the file '834_out.csv' should appear in the C:\chiapas directory, which is a CGIF Flat representation of the 834 sample file used in the example.

Normally, Workflows will start with a CGateArg activity to load some local variables from the automation environment, and then use these variables to do some processing. There are twelve workflows included with the distribution (see the Quick Start guide for more information) that can be used as examples for how to process a variety of situations. Furthermore, there are a number of additional activities to handle tasks such as Email notifications and SecureFTP transfers.

Variables

To create a new variable, first click on the CGateContext activity and then click on the blue Variables label at the bottom area of the screen; this will extend an area showing all the currently defined variables for this scope. Click on the *Create Variable* label and set the name and type. This variable can then be used in your workflows.

Arguments are not currently used in this implementation.

Imports

Like any .NET language, you will need to add a reference to a .NET class assembly to use certain functions. This is accessible via the Imports tab next to the Variables tab, and then either typing in the assembly name or selecting it in the pane. The System.IO assembly needs special attention, as it adds a lot of functionality to a workflow. It is beyond the scope of this manual to detail exactly how .NET assemblies work, but here are a few example System.IO functions that are useful in workflows:

Path.GetFileName(<variable>) – Removes the path information from a given path stored in a string variable

Path.GetFileNameWithoutExtension(<variable>) – Gets only the filename without the extension from a path given in a string variable

SCHEDULER INTERFACE

This interface is dedicated to creating and maintaining the Triggers that underlay the automation system. It is capable of displaying the execution history as well as the message log trail of each event that has been executed. Furthermore, it allows triggers to be set up with the following criteria:

- Run workflow at a set poll interval
- Run workflow at a scheduled poll period on specific days of the week, date ranges or months
- Run workflow in response to new files appearing in an archive directory
- Run workflow in response to files appearing to an upload directory, where they are then moved to a different directory and processed (with an optional file renaming function)
- Run workflow in response to new files appearing in a remote SecureFTP folder

These triggers are maintained as part of the automation system as described in the Workflow Interface section and detailed in the Technical Reference section.

To switch to the Scheduler Interface, launch ChiapasStudio (C:\chiapas\bin\chiapasstudio.exe), pull down the Window menu option and select 'Scheduler Interface'.

A note about polls: The ChiapasService polls the chiapasData database for active triggers every second, and has an internal log it uses to keep track of the last time every active trigger was polled. When the Poll Interval (in seconds) has been exceeded, it will proceed to check the poll.

Active Triggers

All Triggers

Pending Events

TSRC_ID	TSRC_TYPE	TSRC_NAME	TSRC_WF_FILENAME
10	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_010_EncodeFromXML	C:\chiapas\xaml\Pipeline_010_EncodeFromXML.xml
9	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_009_EncodeFromCSV	C:\chiapas\xaml\Pipeline_009_EncodeFromCSV.xml
8	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_008_DecodeToINDB_ForceMerge	C:\chiapas\xaml\Pipeline_008_DecodeToINDB_ForceMerge.xml
7	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_007_DecodeToINDB_Merge	C:\chiapas\xaml\Pipeline_007_DecodeToINDB_Merge.xml
6	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_006_DecodeToXMLDB	C:\chiapas\xaml\Pipeline_006_DecodeToXMLDB.xml
5	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_005_DecodeToXML	C:\chiapas\xaml\Pipeline_005_DecodeToXML.xml
4	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_004_DecodeToCSV	C:\chiapas\xaml\Pipeline_004_DecodeToCSV.xml
3	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_003_ScrubPHI	C:\chiapas\xaml\Pipeline_003_ScrubPHI.xml
2	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_002_CullLF	C:\chiapas\xaml\Pipeline_002_CullLF.xml
1	LOCAL_UPLOAD_DIRECTORY_EVENT	Pipeline_001_SetLF	C:\chiapas\xaml\Pipeline_001_SetLF.xml

Trigger Info

Polling Info

Schedule Info

Event History

New

Delete

Save

Name

Pipeline_001_SetLF

☒ Enabled

Poll Interval

15

Workflow File

C:\chiapas\xaml\Pipeline_001_SetLF.xml

The main interface is divided into three areas. The main interface elements are in the bottom left quadrant of the screen. At the top left is a list showing Active Triggers, All Triggers and Pending Events. On the right is an area reserved for showing message logs from selected events.

Trigger Info Tab

Name – This is the name and/or description of the Trigger.

Enabled – This determines if a trigger is actively polled or not.

Poll Interval – This is the time in seconds to wait between polls to this trigger. Setting this value very low (5 or 15) will mean that this poll will be checked very often. The user should balance between responsiveness from the automation system and excessive resource usage by the background service – generally values of 15 to 30 are best.

Workflow File – This is the XAML workflow file to run when the trigger is fired. Double-clicking into the window will bring up an Open File dialog.

New – Pressing this will create a new, inactive trigger with a default name. In order to change the information from the defaults, click on it in the All Triggers tab at top, enter in the correct information and then press SAVE.

Delete – This allows you to delete an existing trigger from the automation system. The XAML workflow file will not be affected.

Save – This button commits any changes made to the trigger information to the backend database. Any changes made to the trigger will not be saved until the Save button is pressed.

Polling Info Tab

The screenshot shows the 'Polling Info' tab selected in a software interface. At the top, there are four tabs: 'Trigger Info', 'Polling Info', 'Schedule Info', and 'Event History'. To the right of these tabs are three buttons: 'New', 'Delete', and 'Save'. The 'Polling Info' tab contains four radio button options for different poll types: 'Scheduled Poll', 'SQL Triggered Poll', 'Local Upload File Poll', and 'Local Archive File Poll'. Each option has associated input fields. 'Scheduled Poll' has 'Data01' and 'Data02' fields. 'SQL Triggered Poll' has 'Cnn String', 'Query', and 'DB Type' (a dropdown menu) fields. 'Local Upload File Poll' has 'Local UL Dir', 'Dest Dir', and a 'Timestamp' checkbox. 'Local Archive File Poll' has an 'Archive Dir' field.

Chiapas EDI Enterprise supports several trigger types:

Scheduled Poll – When this is selected the Schedule Info tab is enabled and you are able to select a custom polling schedule for this trigger that can make it run at different intervals and at different dates. More information on this is covered in the Schedule Info Tab section.

SQL Triggered Poll - This polling mechanism allows you to drive workflow execution from a database query. If a '1' result is returned from the query, the trigger is fired. It requires the following information (do not use quotes when entering information to the these text boxes):

Cnn String – Database connection string. This is a database connection string with which to run the specified query.

Query – Query to run. An expression like 'SELECT 1' will always fire the trigger, whereas an expression like 'SELECT 0' never will.

DB Type – Database type.

Local Upload File Poll – ChiapasService will scan the specified directory for any files. If any files are found, they are immediately moved to the Destination directory, optionally renamed, and then the trigger is executed with the new path and filename passed as an argument.

Local UL Dir – The source directory to look for new files.

Dest Dir – The destination directory to place the new file. If the file already exists or ChiapasService is unable to move the file to the new location, the event will not fire.

Timestamp – Setting this option will rename the file to include a fourteen digit timestamp in the format <year><month><day><hour><minute><second> when it gets moved to the Destination Directory:

A file named Filename.txt would become:

filename_20140626200801.txt

Note: The trigger is only fired when and if the file can be successfully moved to the Destination directory. If another process is uploading a new file and it is 'locked', it will not be touched by the automation system until the file has completed transferring.

Local Archive File Poll – The background service will poll the directory specified in the Archive Directory textbox every polling interval and compare the files with the entries for this trigger in the FSTOR_FILES table in the chiapasData database. Any new files found will trigger a new event and also insert a new entry in the FSTOR_FILES table. Note that this directory should not accumulate too many files – more than a thousand files could impact the functionality of the Automation service.

Note: In contrast to the Local Upload File Poll, partially downloaded files (such as ending in a '.part') extension *will* fire this trigger. For this reason, only fully completed files should be moved to an archive triggered directory, or an activity at the beginning of the workflow should check the filename for validity.

Schedule Info Tab

This allows you to set the trigger schedule on a specific schedule, repeating at a set interval of seconds, minutes or hour. Furthermore, you can filter the schedule by Day of Week, specific months, and start/end times for daily schedules and begin/end dates for longer term schedules. All of these are treated as conditions that are individually checked during the main workflow trigger timer.

When the event is triggered, the current timestamp is placed in the TSRC_DATA01 field within the row for that trigger. If this field is a null or empty string, then the automation will try to trigger the event at the next polling interval that matches all of the criteria specified in the window.

The screenshot shows the 'Schedule Info' tab in a software interface. It includes buttons for 'New', 'Delete', and 'Save'. Under 'Interval Trigger Types', 'Hourly Interval' is selected. The 'Interval' is set to 1. 'Start Time' is 10:48:00 AM and 'End Time' is 11:48:00 AM. Under 'Day of Week', Monday through Thursday are selected. The 'Date Span - From Date / To Date' section shows two calendar views for June 2014. The 'Months' section has checkboxes for all months from January to December.

Here is a breakdown of the various schedule filters. In order for the event to fire, all options that are filled in will need to be true for the event to fire.

Run Always – Run every time the trigger is polled

Second Interval – Run after the number of seconds specified in Interval has elapsed

Minute Interval – Run after the number of

minutes specified in Interval has elapsed

Hourly Interval – Run after the number of hours specified in Interval has elapsed

Start Time / End Time – Specify a window in a 24-hour clock when the event can fire

Date Span – From Date / To Date – The Calendar dialogs will specify a date span window during which the event can fire

Day Of Week – Fire the event only on specified days of the week

Months – Fire the event only on specified months

Example: To set an event to happen once a day at 8 AM, set the Trigger Type to Hourly Interval, set an Interval to 1 and set the Start Time to 8:00 AM and the End Time to 8:55 AM. Even if the ChiapasService was suspended for a half an hour from 8 to 8:30 AM, the event would still fire at 8:31 AM and then again at 8 AM the following day.

Note: The Automation system does not use the current timestamp when updating the Last Execution on the TSRC table, as that would introduce increasing inaccuracies. Instead, it applies the Interval in seconds, minutes or hours to the last execution time and then generates a new event. If you set an Interval of 1 second and the Trigger Poll at 30 seconds, then the automation system would be constantly trying to 'catch up' and get further behind. Therefore, it's important to be aware of how the system works so that the values can be set correctly.

Note: All workflows are executed concurrently unless a database value is changed on the TSRC table to enabled serial execution (TSRC_DATA05). There is no way to set this directly from the interface.

Events History Tab

This shows the trigger history. Clicking on one of the events will populate the log screen on the right, which shows all of the messages generated from the Workflow during the execution of the event. This will allow you to easily see the reasons behind any Critical Errors that may have occurred during workflow execution.

TECHNICAL REFERENCE

This part of the manual is dedicated to explaining the automation aspects of the platform, including:

- Core Database Schema
- Workflow Activities

The Core Database Schema describes the structure and function of all objects within the chiapasData distribution database, and how they are used in conjunction with the Automation system to encode, decode and store EDI information. The SQL database views used to generate the Seed files are also described in-depth.

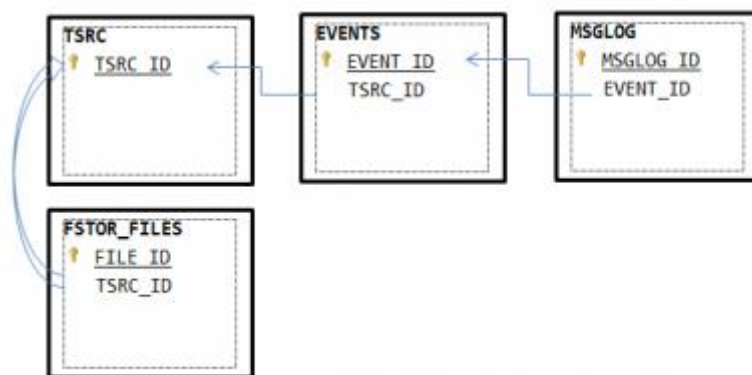
The Workflow Activities section includes all of the Chiapas EDI Enterprise native Activities included with the distribution. Windows Workflow Foundation 4 is a Microsoft technology oriented towards designing processes using a graphical user environment; Chiapas EDI Enterprise contains all of the tools needed to create and maintain these processes. It should be noted that certain WWF 4 features such as Workflow Persistence and bookmarking, are not supported.

CORE DATABASE SCHEMA

When the background automation service 'ChiapasService' first starts up, it loads the contents of the file C:\chiapas\bin\cnstr.txt and uses that as the .NET connection string to the chiapasData database. This database has several purposes:

- Store all automation objects, including triggers, events, messages and the filename pool used to find new files appearing in an Archive trigger directory
- Provide the infrastructure necessary for the built-in automation scripts for storing or retrieving supported HIPAA EDI transactions. This includes all of the BIN* tables as well as all of the stored procedures.
- Provides the objects necessary for creating compliant EDI transactions from mock-up backend data sources. This includes sample data tables holding completely fictitious subscribers, claims, services and providers. Data is present to create compliant transactions for twelve EDI transactions using only a minimal set of mapped elements.

Automation Schema



Note: In the relationship diagram above, the underlined column name is the primary key while the other fields represent foreign keys.

TSRC Table

This table is responsible for storing active and inactive triggers. The different types of triggers are covered in the chapter on the Scheduling System.

Column Name	Data Type	Purpose
TSRC_ID	INT, PK	Primary Key (Identity)
TSRC_ENABLED	INT	0 = Trigger Disabled; 1 = Trigger Enabled
TSRC_TYPE	VARCHAR(50)	Indicates the type of trigger. Valid values: SCHEDULED_EVENT SQL_TRIGGERED_EVENT LOCAL_UPLOAD_DIRECTORY_EVENT LOCAL_ARCHIVE_DIRECTORY_EVENT
TSRC_WF_FILENAME	VARCHAR(250)	Full path to the Workflow .XAML file that is executed as part of the event spawned by this trigger
TSRC_POLL_INTERVAL	INT	This is the number of seconds the background ChiapasService will wait between checks for this trigger. Caution should be exercised when setting this to a value less than 60. This number of seconds should always be <i>less</i> than the elapsed time specified by the interval within a Scheduled Event.
TSRC_NAME	VARCHAR(50)	Name of the trigger
TSRC_DATA01	VARCHAR(4000)	Trigger Data 01. This value varies according to the Trigger type: SCHEDULED_EVENT: Last Execution Time. SQL_TRIGGERED_EVENT: SQL Connection String. LOCAL_UPLOAD_DIRECTORY_EVENT: Polling directory for files LOCAL_ARCHIVE_DIRECTORY_EVENT: Polling directory for new files
TSRC_DATA02	VARCHAR(4000)	Trigger Data 02. This value varies according to the Trigger type: SCHEDULED_EVENT: XML representation of the Schedule dialog. SQL_TRIGGERED_EVENT: Database type. LOCAL_UPLOAD_DIRECTORY_EVENT: Directory to place uploaded files prior to processing
TSRC_DATA03	VARCHAR(4000)	Trigger Data 03. SQL_TRIGGERED_EVENT: SQL to run against the specified database connection; a result of Integer '1' or VARCHAR '1' will fire the trigger.
TSRC_DATA04	VARCHAR(4000)	Trigger Data 04. Reserved for future expansion.
TSRC_DATA05	VARCHAR(4000)	Trigger Data 05. Setting this to the value 'SERIAL' will prevent concurrent Workflow execution; a new workflow instance will only be launched if the previous instance has completed execution.
TSRC_CREATEDAT	DATETIME	Timestamp when the trigger was created

EVENTS Table

This table stores events as they occur across various stages, and are the mechanism by which the background automation Worker Processes are set to execute workflows. There are three methods to execute a Workflow:

1. Set a trigger on the Scheduler. The scheduler can execute a workflow to run according to four triggering criteria: SQL result, timed and scheduled interval, and both 'Upload' and 'Archive' file trigger mechanisms
2. Schedule a SQL Server Job to insert a new row to the EVENTS table
3. Manually insert a new row to the EVENTS table

In every three of the above methods, new rows are created with the EVENT_OWNER column set to -1. This is the signal to Worker Processes polling the chiapasData database in the background that this is event has yet to be processed. When a Worker Process is idling, it will look for these events and attempt to take ownership; if this succeeds, the process will load the XAML Workflow file and process it with the arguments provided within the EVENTS entry.

During Workflow execution, the CGateArgs command acts to assign the four VARCHAR EVENT_DATA columns to string variables. Also, the workflow can also obtain the EVENT_ID by calling on CGateParse with the 'EVENT_ID' string so that it can obtain information about the activating trigger.

Once the workflow is complete, any messages residing in the *state* object are sent to the MSGLOG table. If an unhandled exception occurs, then the state object will be unavailable - the EVENT_RESULT column is set to 'THREAD ABORT ERROR - SEE C:\CHIAPAS\ERRLOG FILE' and the user must research the file created within the C:\chiapas directory that contains more information about the exception.

Column Name	Data Type	Purpose
EVENT_ID	INT, PK	Primary Key (Identity)
EVENT_OWNER	INT	This is refers to the numeric ID of the background worker process that took ownership and processed the event. For the lowest licensing tier which supports two Worker Processes, this number will be either 1 or 2. Events that are not assigned to an owner will start with -1. All events inserted into this table should have a -1 in the column or else it will not be processed.
TSRC_ID	INT	Foreign key to the Trigger table of the trigger that spawned this event.
EVENT_DATA01	VARCHAR(200)	Event Data 01. For Scheduled and SQL Triggered events that do not carry data, this value is blank. For Archive events, this value is the filename <i>without</i> path. For Upload events, this value is the filename <i>with</i> path.
EVENT_DATA02	VARCHAR(200)	Event Data 02.
EVENT_DATA03	VARCHAR(200)	Event Data 03.
EVENT_DATA04	VARCHAR(200)	Event Data 04.
EVENT_CREATED	DATETIME	Timestamp of when the event was created
EVENT_PROCESS_BEGIN	DATETIME	Timestamp of when the event began processing
EVENT_PROCESS_COMPLETE	DATETIME	Timestamp of the completion of processing
EVENT_RESULT	VARCHAR(200)	Values: 'CRITICAL ERROR, SEE MESSAGE LOGS', 'CRITICAL ERROR, NO LOGS', 'COMPLETE WITH MESSAGE LOGS', 'COMPLETE, NO LOGS'

MSGLOG Table

This table stores the Message Logs that are associated with a specific event. They are created by the activities within a Chiapas EDI Enterprise workflow, and the possible error messages are documented within each activity.

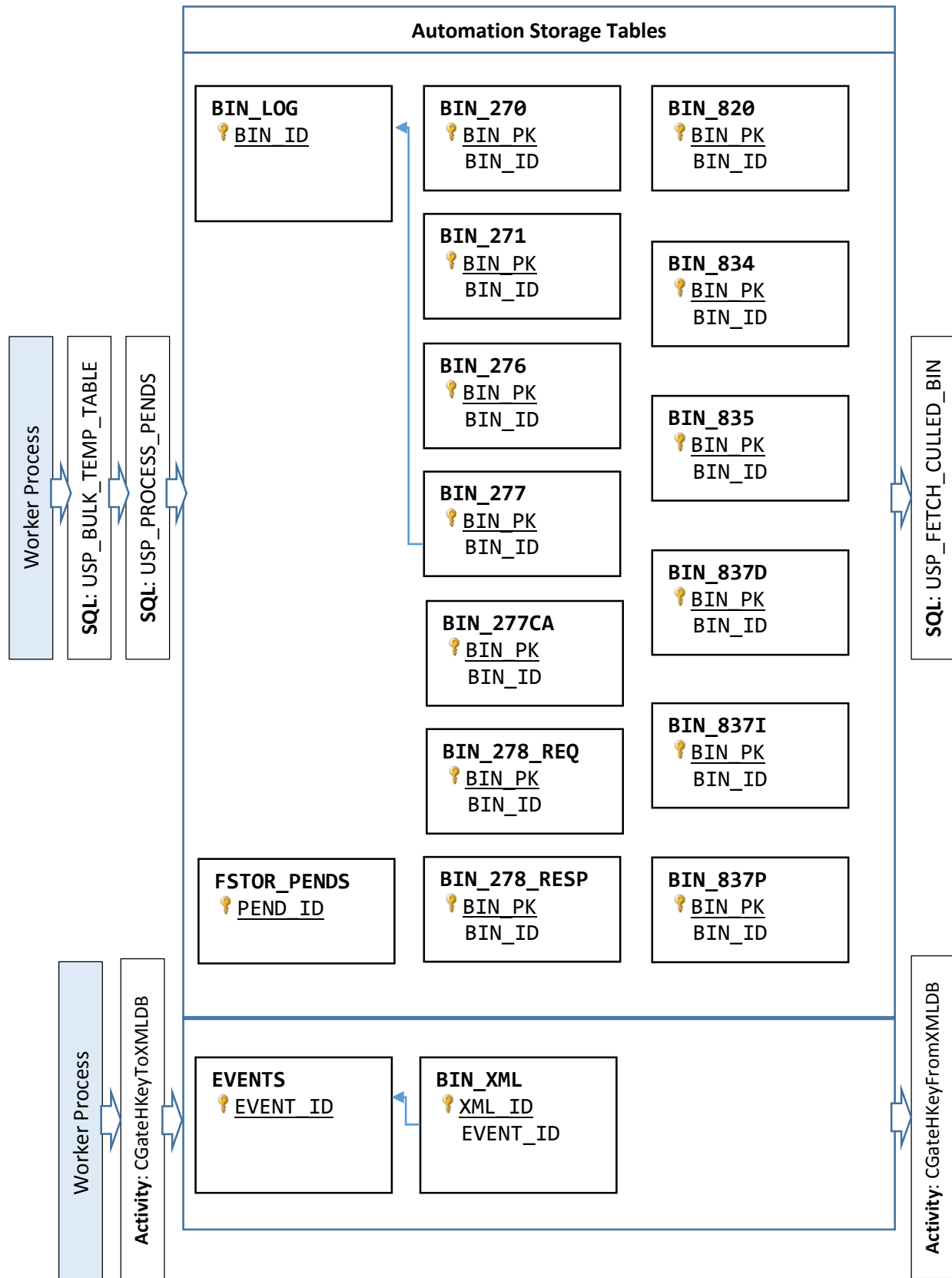
Column Name	Data Type	Purpose
MSGLOG_ID	INT, PK	Primary Key (Identity)
Origin	VARCHAR(200)	Chiapas EDI Module that created the message
Message	VARCHAR(200)	Error Message
MsgData_Str	VARCHAR(200)	Message Data - String
MsgData_Num1	VARCHAR(200)	Message Data - Num 1
MsgData_Num2	VARCHAR(200)	Message Data - Num 2
IsError	VARCHAR(200)	Flags whether this message is an error
TimeStamp	VARCHAR(200)	Timestamp of when the error was generated
EVENT_ID	INT	ID of the triggering Event
MSGLOG_NR	VARCHAR(200)	Number indicating the sequence of the message within the event

FSTOR_FILES Table

This table stores filenames associated with an archive directory. An archive directory is defined as a holding area that holds a number of uniquely named files. As new files appear in that directory, the automation system will trigger a workflow to act upon that file. This table assists the automation system in deciding when a new file arrives. At specific interval, all of the files in a directory are compared with entries in this directory. Note that this gets increasingly inefficient the more files accumulate in a directory; it is recommended that files should be archived so that the total number of files in a directory is less than 1000.

Column Name	Data Type	Purpose
FILE_ID	INT, PK	Primary Key (Identity)
TSRC_ID	INT	Foreign key to the TSRC row associated with an archive trigger
FILE_NM	VARCHAR(250)	File name of the file. Note that as the trigger is linked to a specific directory, this filename does <i>not</i> contain path information.

AUTOMATION STORAGE SCHEMA



The Automation Storage Schema is described on the previous page. This storage system has the capability to store supported EDI files in two different formats: Flat table storage and XML. Here is a comparison of both:

Flat

- Easiest to use; usable for a majority of business situations
- Not space efficient; results in large duplication of information within the SQL tables

XML

- Most difficult to use; provides the most flexibility
- Space efficient - file size is usually only 2-3 times the original EDI file

Flat BIN Storage

In the previous diagram, the upper part is dedicated to the Input/Storage/Output flow of the Flat BIN storage objects. Data is inserted via the **USP_BULK_TEMP_TABLE** and **USP_PROCESS_PENDS** stored procedures, as called by the following workflows:

Pipeline_007_DecompileToBINDB_Merge

Pipeline_008_DecompileToBINDB_ForceMerge

Pipeline_ProcessPends

For every file stored, a new BIN_LOG row is created that gives that particular file a primary key (BIN_ID) and associates a filename, specification type and the EVENT_ID that spawned this entry. The BIN_ID is then stored as a foreign key in each of the BIN_x tables. Data is imported by two stored procedures working in tandem: USP_BULK_TEMP_TABLE followed by USP_PROCESS_PENDS. USP_BULK_TEMP_TABLE creates a temporary table that is populated by the workflow. It will also create a new entry in the FSTOR_PENDS table, where a separate SQL-driven event will later transfer the temp table to the destination table via the USP_PROCESS_PENDS stored procedure.

The reason why the workflows do not directly insert the data into the destination tables is one of resource contention: under certain heavy load conditions with many workflows inserting data into the same destination table, SQL Server will timeout or 'deadlock' some of the processes. There is no issue with SQL Server accepting data from many dozens of sources into temp tables, however, so this method is used to increase the reliability of the automation environment.

The **Merge** workflow will pull in all data from the incoming file that matches the mappings present within the BIN table. Elements that are present in the incoming file and not mapped within the receiving BIN table will be flagged as an error message. This can be thought of as 'lossy' EDI storage, where the emphasis is on a fixed table schema as opposed to guaranteeing all of the incoming fields will be stored.

The **ForceMerge** workflow is similar to the Merge workflow, but is also focused on making sure all data is captured. When an incoming file presents new mappings that are not stored in the destination tables, the USP_PROCESS_PENDS stored procedure will add these new fields to the destination table. Note that this workflow is not appropriate for certain 835 files having a more than a dozen PLB segments – the large amount of mappings they generate can overload the architectural limits of SQL Server.

The **USP_FETCH_CULLED_BIN** stored procedure is the primary way to extract CGIF Flat tables out of the BIN tables. It returns a recordset that meets the following conditions:

- Only the columns that were originally present within the file should be returned. Since the same BIN table could be used for both 4010 and 5010 files, it is responsible for sending only the columns that actually stored data for that BIN_ID.
- The specification tag should always be the first column returned
- The rows should be in a consistent order – for example, claims and service lines should be aggregated together properly. Since SQL Views are not engineered to obey ORDER BY clauses, a stored procedure must be used to return the dataset in proper order.

The **USP_ORDEREDI** is alternate method of generating properly sorted CGIF Flat data, but is not plugged directly into the BIN system. Instead, this procedure represents a way of guaranteeing proper sorting of CGIF Flat data sources, such as the vw_Export views included in the database. SQL Server views cannot order the resultset unless the ORDER BY clause is external to the view (for example, SELECT * FROM vw_Export_834 ORDER BY L2000_S02_REF02_SUB_ID_OVL_SUB_NR), but stored procedures can guarantee proper order. This stored procedure will generate an ORDER BY clause suitable for the transaction being extracted and return the data in sorted order.

XML Storage

In this system, Chiapas EDI Enterprise workflows store data directly to SQL tables as CGIFXML2 formatted rows. The XML can be queried directly by SQL Server, or another workflow can regenerate the EDI file from the XML storage bin. The BIN_XML system does not utilize the BIN_LOG table, but instead

The workflow that decodes files into the BIN_XML is **Pipeline_006_DecompileToXMLDB**. It will automatically insert the correct EVENT_ID into BIN_XML table along with the CGIFXML2 rendition of the EDI file. To retrieve this data, the manually triggered workflow **Pipeline_011_EncodeFromXMLDB** will create new EDI files from the BIN_XML table.

FSTOR_PENDS Table

This table helps orchestrate the 'delayed write' architecture behind the two Flat Table Decoder Pipelines. This architecture ensures multiple Workers are never trying to insert into the same SQL database table by sending all data to temporary tables and having a single serial pipeline handle the actual final data insertions. This table records the 'Source' temp table and the 'Destination' production table where the data will be shuttled.

Column Name	Data Type	Purpose
PEND_ID	INT, PK	Primary Key (Identity)
EVENT_ID	INT	Foreign key to the EVENT table of the event that created this entry
BIN_ID	INT	Foreign key to the BIN_LOG table for the specific file being handled
TEMP_TBL	VARCHAR(200)	Temporary Table name – generally, it will called 'TEMP' followed by the EVENT_ID. This table will get dropped once it is successfully processed.
DEST_TBL	VARCHAR(200)	Destination Table name – The name of the destination table located on the chiapasData database. If the temp table has fields not present in the destination table and this is a ForceMerge workflow, the PROCESS PENDS procedure will add the missing fields to the destination table.
CREATE_DATE	DATETIME	Timestamp when this table entry was created.
FINISH_DATE	DATETIME	Timestamp when this table entry was completed.
STATUS	VARCHAR(50)	<p>This field represents the status of the Delayed Write operation.</p> <p>NEW – This is the initial status of the operation where the temporary table is created but not yet populated with data.</p> <p>READY – This is the status of a Merge Decode workflow, and signifies that the temporary table has been populated with data and is ready to be inserted back to the destination table. On completion, the status is updated to DROP.</p> <p>FORCE – This is the status of a ForceMerge Decode workflow, and signifies that the temporary table has been populated with data. If there are fields present within the TEMP table missing in the destination table, they will be added in prior to the insert operation. On completion, the status is updated to DROP.</p> <p>DROP – This tells the automation system that the data has been inserted and the temp table can be dropped. Once this is done, the status changes to DONE.</p> <p>DONE – This signifies the operation is fully complete.</p>

BIN_LOG Table

This table acts as an index to the other thirteen tables in the CGIF Flat storage system. The BIN_ID primary key is a foreign key to the other tables.

Column Name	Data Type	Purpose
BIN_ID	INT, PK	Primary Key (Identity)
BIN_FILENAME	VARCHAR(300)	EDI Filename
BIN_RECV_DATE	DATETIME	Timestamp when the log entry was created
BIN_TYPE	VARCHAR(20)	
EVENT_ID	INT	Foreign key to the EVENTS table for the reference to the event that spawned this BIN_LOG record

BIN_270 Table

This BIN table holds mappings matching that of a minimal 5010 270 A1 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
M1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_PASSWD		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		
GSHDR_S01_GS08_VERS_ID_CD		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S02_BHT01_STRUCTR_CD		
STHDR_S02_BHT02_TS_PURPS_CD		
STHDR_S02_BHT03_SUBMTR_TRANSCTN_ID		

STHDR_S02_BHT04_TS_CREATN_DT		
STHDR_S02_BHT05_TS_CREATN_TIM		
STHDR_S04_SE02_TCN		
L2000A_S01_HL01_HIERCHCL_ID_NR		
L2100A_PR_S01_NM103_NFO_SOURC_LAST_ORG_NM_OVL_NONPRSN_ENTY		
L2100A_PR_S01_NM109_NFO_SOURC_PRIMARY_ID_OVL_FED_TAX_ID_NR		
L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2100B_1P_S01_NM103_NFO_RECVR_LAST_ORG_NM_OVL_NONPRSN_ENTY		
L2100B_1P_S01_NM109_NFO_RECVR_ID_NR_OVL_FED_TAX_ID_NR		
L2000C_S01_HL01_HIERCHCL_ID_NR		
L2000C_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000C_S01_HL04_HIERCHCL_CHILD_CD		
L2100C_S01_NM102_ENTY_TYPE_QUAL		
L2100C_S01_NM103_SUB_LNAME		
L2100C_S01_NM104_SUB_FNAME		
L2100C_S01_NM109_PRIMARY_ID_OVL_MEM_ID_NR		
L2100C_S06_DMG02_BIRTH_DT_OVL_DT_CCYYMMDD		
L2100C_S06_DMG03_GENDR_CD		
L2110C_S01_EQ01_SVC_TYPE_CD		

BIN_271 Table

This BIN table holds mappings matching that of a minimal 5010 271 A1 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
N1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		
GSHDR_S01_GS08_VERS_ID_CD		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S02_BHT01_STRUCTR_CD		
STHDR_S02_BHT02_TS_PURPS_CD		
STHDR_S02_BHT03_SUBMTR_TRANSCTN_ID		

STHDR_S02_BHT04_TS_CREATN_DT		
STHDR_S02_BHT05_TS_CREATN_TIM		
STHDR_S04_SE02_TCN		
L2000A_S01_HL01_HIERCHCL_ID_NR		
L2000A_S01_HL04_HIERCHCL_CHILD_CD		
L2100A_PR_S01_NM103_NFO_SOURC_LAST_ORG_NM_OVL_NONPRSN_ENTY		
L2100A_PR_S01_NM109_NFO_SOURC_PRIMARY_ID_OVL_FED_TAX_ID_NR		
L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000B_S01_HL04_HIERCHCL_CHILD_CD		
L2100B_1P_S01_NM102_ENTY_TYPE_QUAL		
L2100B_1P_S01_NM103_NFO_RECVR_LAST_ORG_NM		
L2100B_1P_S01_NM109_NFO_RECVR_ID_NR_OVL_FED_TAX_ID_NR		
L2000C_S01_HL01_HIERCHCL_ID_NR		
L2000C_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000C_S01_HL04_HIERCHCL_CHILD_CD		
L2100C_S01_NM103_SUB_LNAME		
L2100C_S01_NM104_SUB_FNAME		
L2100C_S01_NM109_PRIMARY_ID_OVL_MEM_ID_NR		
L2100C_S03_N301_ADRS_LIN		
L2100C_S04_N401_CITY_NM		
L2100C_S04_N402_STAT_CD		
L2100C_S04_N403_SUB_POSTL_ZON_ZIP_CD		
L2100C_S10_DTP03_DT_TM_PERD_OVL_PLAN_D8		
L2100C_S10_DTP03_I02_DT_TM_PERD_OVL_ELIG_BEGN_D8		
L2110C_S01_EB01_BENFT_NFO		
L2110C_S01_EB03_SVC_TYPE_CD		

BIN_276 Table

This BIN table holds mappings matching that of a minimal 5010 276 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
OO_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S01_ST03_VERS_RELS_INDSTRY_ID		
STHDR_S02_BHT01_STRUCTR_CD		
STHDR_S02_BHT03_REF_ID		
STHDR_S02_BHT04_TS_CREATN_DT		

STHDR_S02_BHT05_TS_CREATN_TIM		
STHDR_S04_SE02_TCN		
L2000A_S01_HL01_HIERCHCL_ID_NR		
L2100A_S01_NM103_PAYR_NM_OVL_NONPRSN_ENTY		
L2100A_S01_NM109_PAYR_ID_OVL_PAYR_ID		
L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2100B_S01_NM102_ENTY_TYPE_QUAL		
L2100B_S01_NM103_NFO_RECVR_LAST_ORG_NM		
L2100B_S01_NM109_NFO_RECVR_ID_NR_OVL_ETN_NR		
L2000C_S01_HL01_HIERCHCL_ID_NR		
L2000C_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2100C_S01_NM102_ENTY_TYPE_QUAL		
L2100C_S01_NM103_PROV_LAST_ORG_NM		
L2100C_S01_NM109_PROV_ID_OVL_FED_TAX_ID_NR		
L2000D_S01_HL01_HIERCHCL_ID_NR		
L2000D_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000D_S01_HL04_HIERCHCL_CHILD_CD		
L2000D_S02_DMG02_BIRTH_DT_OVL_DT_CCYYMMDD		
L2000D_S02_DMG03_GENDR_CD		
L2100D_S01_NM103_SUB_LNAME_OVL_PERSN		
L2100D_S01_NM104_SUB_FNAME		
L2100D_S01_NM109_SUB_ID_OVL_MEM_ID_NR		
L2200D_S01_TRN02_CURNT_TRANSCTN_TRAC_NR		
L2200D_S02_REF02_PAYR_CLM_CONTRL_NR_OVL_PAYR_CLM_NR		
L2200D_S10_DTP03_SVC_PERD_OVL_SVC_RD8		

BIN_277 Table

This BIN table holds mappings matching that of a minimal 5010 277 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
P0_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S01_ST03_VERS_RELS_INDSTRY_ID		
STHDR_S02_BHT01_STRUCTR_CD		
STHDR_S02_BHT03_ORGNTR_APLCTN_TRANSCTN_ID		
STHDR_S02_BHT04_TS_CREATN_DT		

STHDR_S02_BHT05_TS_CREATN_TIM		
STHDR_S04_SE02_TCN		
L2000A_S01_HL01_HIERCHCL_ID_NR		
L2100A_S01_NM103_PAYR_NM_OVL_NONPRSN_ENTY		
L2100A_S01_NM109_PAYR_ID_OVL_PAYR_ID		
L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000B_S01_HL04_HIERCHCL_CHILD_CD		
L2100B_S01_NM102_ENTY_TYPE_QUAL		
L2100B_S01_NM103_NFO_RECVR_LAST_ORG_NM		
L2100B_S01_NM109_NFO_RECVR_ID_NR_OVL_ETN_NR		
L2000C_S01_HL01_HIERCHCL_ID_NR		
L2000C_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000C_S01_HL04_HIERCHCL_CHILD_CD		
L2100C_S01_NM102_ENTY_TYPE_QUAL		
L2100C_S01_NM103_PROV_LAST_ORG_NM		
L2100C_S01_NM109_PROV_ID_OVL_FED_TAX_ID_NR		
L2000D_S01_HL01_HIERCHCL_ID_NR		
L2000D_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000D_S01_HL04_HIERCHCL_CHILD_CD		
L2100D_S01_NM103_SUB_LNAME_OVL_PERSN		
L2100D_S01_NM104_SUB_FNAME		
L2100D_S01_NM109_SUB_ID_OVL_MEM_ID_NR		
L2200D_S01_TRN02_REFNCD_TRANSCTN_TRAC_NR		
L2200D_S02_STC01_01_HLTHCR_CLM_STATS_CATGRY_CD		
L2200D_S02_STC01_02_STATS_CD		
L2200D_S02_STC02_STATS_NFO_EFF_DT		
L2200D_S03_REF02_PAYR_CLM_CONTRL_NR_OVL_PAYR_CLM_NR		
L2200D_S09_DTP03_SVC_PERD_OVL_SVC_RD8		

BIN 277CA Table

This BIN table holds mappings matching that of a minimal 5010 277CA EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
P5_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S02_BHT03_REF_ID		
STHDR_S02_BHT04_TS_CREATN_DT		
STHDR_S02_BHT05_TS_CREATN_TIM		
STHDR_S06_SE02_TCN		

L2000A_S01_HL01_HIERCHCL_ID_NR		
L2100A_PR_S01_NM103_NFO_SOURC_LAST_ORG_NM_OVL_NON_PERSN_ENTY		
L2100A_PR_S01_NM109_NFO_SOURC_PRIMARY_ID_OVL_PAYR_ID		
L2200A_S01_TRN02_NFO_SRC_APPLCTN_TRAC_ID		
L2200A_S02_DTP03_DT_TM_PERD_OVL_RECVD_D8		
L2200A_S03_DTP03_NFO_SRC_PROCSS_DT_OVL_PROCSS_D8		
L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000B_S01_HL04_HIERCHCL_CHILD_CD		
L2100B_S01_NM103_NFO_RECVR_LAST_ORG_NM_OVL_NON_PERSN_ENTY		
L2100B_S01_NM109_NFO_SOURC_PRIMARY_ID_OVL_ETIN		
L2200B_S01_TRN02_CLM_TRANSCTN_BATCH_NR		
L2200B_S02_STC01_01_HLTH_CR_CLM_STATS_CAT_CD		
L2200B_S02_STC01_02_HLTH_CR_CLM_STATS_CD		
L2200B_S02_STC02_STATS_NFO_EFF_DT		
L2200B_S02_STC03_ACTN_CD		
L2200B_S02_STC04_TOTL_SUBMTD_CHGS_WORK		
L2200B_S03_QTY02_TOTL_ACCPTD_QTY		
L2200B_S04_QTY02_TOTL_REJECTD_QTY		

BIN 278 REQ Table

This BIN table holds mappings matching that of a minimal 5010 278 Request EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
Q0_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S01_ST03_IMPLMNTN_GUID_VERSN_NM		
STHDR_S02_BHT01_STRUCTR_CD		

STHDR_S02_BHT02_TS_PURPS_CD		
STHDR_S02_BHT03_SUBMTR_TRANSCTN_ID		
STHDR_S02_BHT04_TS_CREATN_DT		
STHDR_S02_BHT05_TS_CREATN_TIM		
STHDR_S04_SE02_TCN		
L2000A_S01_HL01_HIERCHCL_ID_NR		
L2010A_X3_S01_NM102_ENTY_TYPE_QUAL		
L2010A_X3_S01_NM103_UMO_NM_LAST_ORG_NM		
L2010A_X3_S01_NM109_ID_CD_OVL_EMPLYR_ID_NR		
L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2010B_1P_S01_NM102_ENTY_TYPE_QUAL		
L2010B_1P_S01_NM103_REQSTR_LAST_ORG_NM		
L2010B_1P_S01_NM104_REQSTR_FNAME		
L2010B_1P_S01_NM109_REQSTR_ID_OVL_ETN_NR		
L2000C_S01_HL01_HIERCHCL_ID_NR		
L2000C_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2010C_S01_NM103_SUB_LNAME		
L2010C_S01_NM104_SUB_FNAME		
L2010C_S01_NM109_PRIMARY_ID_OVL_MEM_ID_NR		
L2000E_S01_HL01_HIERCHCL_ID_NR		
L2000E_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000E_S01_HL04_HIERCHCL_CHILD_CD		
L2000E_S03_UM01_REQST_CATGRY_CD		
L2000E_S03_UM02_CERT_TYPE_CD		
L2000E_S03_UM04_01_FACLTY_TYPE_CD		
L2000E_S03_UM04_02_FACLTY_CD_QUAL		
L2000E_S03_UM09_RELS_NFO_CD		
L2000E_S11_DTP03_PROPSD_ACTL_ADMSN_DT_OVL_ADMSN_D8		

BIN_278_RESP Table

This BIN table holds mappings matching that of a minimal 5010 278 Response EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
R0_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S01_ST03_IMPLMNTN_GUID_VERSN_NM		
STHDR_S02_BHT01_STRUCTR_CD		
STHDR_S02_BHT03_SUBMTR_TRANSCTN_ID		
STHDR_S02_BHT04_TS_CREATN_DT		
STHDR_S02_BHT05_TS_CREATN_TIM		

STHDR_S02_BHT06_TRANSCTN_TYPE_CD		
STHDR_S04_SE02_TCN		
L2000A_S01_HL01_HIERCHCL_ID_NR		
L2000A_S01_HL04_HIERCHCL_CHILD_CD		
L2010A_X3_S01_NM102_ENTY_TYPE_QUAL		
L2010A_X3_S01_NM103_UMO_NM_LAST_ORG_NM		
L2010A_X3_S01_NM109_ID_CD_OVL_EMPLYR_ID_NR		
L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000B_S01_HL04_HIERCHCL_CHILD_CD		
L2010B_1P_S01_NM102_ENTY_TYPE_QUAL		
L2010B_1P_S01_NM103_REQSTR_LAST_ORG_NM		
L2010B_1P_S01_NM104_REQSTR_FNAME		
L2010B_1P_S01_NM109_REQSTR_ID_OVL_EMPLYR_ID_NR		
L2000C_S01_HL01_HIERCHCL_ID_NR		
L2000C_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000C_S01_HL04_HIERCHCL_CHILD_CD		
L2010C_S01_NM103_SUB_LNAME		
L2010C_S01_NM104_SUB_FNAME		
L2010C_S01_NM109_PRIMARY_ID_OVL_MEM_ID_NR		
L2010C_S06_DMG02_BIRTH_DT_OVL_DT_CCYYMMDD		
L2010C_S06_DMG03_GENDR_CD		
L2000E_S01_HL01_HIERCHCL_ID_NR		
L2000E_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000E_S01_HL04_HIERCHCL_CHILD_CD		
L2000E_S04_UM01_REQST_CATGRY_CD		
L2000E_S04_UM02_CERT_TYPE_CD		
L2000E_S04_UM03_TYPE_CD		
L2000E_S04_UM04_01_FACLTY_TYPE_CD		
L2000E_S04_UM04_02_FACLTY_CD_QUAL		
L2000E_S05_HCR01_ACTN_CD		
L2000E_S05_HCR02_REVW_ID_NR		

BIN_820 Table

This BIN table holds mappings matching that of a minimal 5010 820 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
S0_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_TS_CONTRL_NR		
STHDR_S01_ST03_IMPLMNTN_CONVNTN_REFNC		
STHDR_S02_BPR02_TOTL_PREM_PMT_AMT_OVL_REMTNC_NFO_ONLY		

STHDR_S02_BPR04_PMT_METHD_CD		
STHDR_S02_BPR10_PAYR_ID		
STHDR_S02_BPR16_CHK_IS_EFT_EFF_DT		
STHDR_S03_TRN01_TYPE_CD		
STHDR_S03_TRN02_CHK_EFT_TRAC_NR		
STHDR_S15_SE02_TCN		
L1000A_S01_N102_LAST_ORG_NM		
L1000A_S01_N104_PREM_RECVR_ID_CD_OVL_FED_TAX_ID_NR		
L1000B_S01_N102_PAYR_NM		
L1000B_S01_N104_PAYR_ID_OVL_FED_TAX_ID_NR		
L2000B_S01_ENT01_ASGND_NR		
L2000B_S01_ENT04_RECVR_INDVDL_ID_OVL_EMPTY_ID_NR		
L2100B_IL_S01_NM103_INDVDL_LNAME		
L2100B_IL_S01_NM104_INDVDL_FNAME		
L2100B_IL_S01_NM109_INDVDL_ID_OVL_INSRD_UNQ_ID_NR		
L2300B_S01_RMR02_INS_REMTNC_REF_NR_OVL_HEALTH_INS_POLCY_NR		
L2300B_S01_RMR04_DETL_PREM_PMT_AMT		

BIN_834 Table

This BIN table holds mappings matching that of a minimal 5010 834 A1 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
T1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_PASSWD		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		

GSHDR_S01_GS08_VERS_ID_CD		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S01_ST03_IMPLMNTN_CONVNTN_REFNC		
STHDR_S02_BGN01_TS_PURPS_CD		
STHDR_S02_BGN02_TS_REF_NR		
STHDR_S02_BGN03_TS_CREATN_DT		
STHDR_S02_BGN04_TS_CREATN_TIM		
STHDR_S02_BGN08_ACTN_CD		
STHDR_S10_SE02_TCN		
L1000A_S01_N102_PLAN_SPONSR_NM		
L1000A_S01_N104_SPONSR_ID_OVL_FED_TAX_ID_NR		
L1000B_S01_N102_INSR_NM		
L1000B_S01_N104_INSR_ID_CD_OVL_FED_TAX_ID_NR		
L2000_S01_INS01_MEM_IND		
L2000_S01_INS02_INDVDL_REL_CD		
L2000_S01_INS03_MAINTNC_TYPE_CD		
L2000_S01_INS05_BENFT_STATS_CD		
L2000_S01_INS08_EMPLYMNT_STATS_CD		
L2000_S02_REF02_SUB_ID_OVL_SUB_NR		
L2000_S03_REF02_MEM_GRP_POLCY_NR_OVL_GRP_POLCY_NR		
L2000_S04_REF02_SUPP_ID_OVL_CLIENT_REPRTNG_CATGRY		
L2000_S05_DTP03_STATS_NFO_EFF_DT_OVL_ELIG_BEGN_D8		

L2100A_IL_S01_NM103_MEM_LNAME_OVL_PERSN		
L2100A_IL_S01_NM104_MEM_FNAME		
L2100A_IL_S02_PER04_COMM_NR_OVL_HOM_PHON_NR		
L2100A_IL_S03_N301_ADRS_LIN		
L2100A_IL_S04_N401_CITY_NM		
L2100A_IL_S04_N402_STAT_CD		
L2100A_IL_S04_N403_MEM_POSTL_ZON_ZIP_CD		
L2100A_IL_S05_DM02_BIRTH_DT_OVL_DT_CCYYMMDD		
L2100A_IL_S05_DM03_GENDR_CD		

BIN_835 Table

This BIN table holds mappings matching that of a minimal 5010 835 A1 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to BIN_LOG
U1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		

GSHDR_S01_GS08_VERS_ID_CD		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S02_BPR02_TOTL_ACTL_PROV_PMT_AMT_OVL_REMTNC_NFO_ONLY		
STHDR_S02_BPR03_CREDIT_DEBT_FLAG_CD		
STHDR_S02_BPR04_PMT_METHD_CD		
STHDR_S02_BPR16_CHK_IS_EFT_EFF_DT		
STHDR_S03_TRN02_CHK_EFT_TRAC_NR		
STHDR_S03_TRN03_PAYR_ID		
STHDR_S05_REF02_RECVR_ID_OVL_RECVR_ID_NR		
STHDR_S07_DTM02_PRODCTN_DT_OVL_PRODCTN		
STHDR_S12_SE02_TCN		
L1000A_S01_N102_PAYR_NM		
L1000A_S01_N104_PAYR_ID_OVL_MDCR_MDCD_SVCS_PLAND		
L1000A_S02_N301_ADRS_LIN		
L1000A_S03_N401_CITY_NM		
L1000A_S03_N402_STAT_CD		
L1000A_S03_N403_PAYR_POSTL_ZON_ZIP_CD		
L1000A_S06_PER02_PAYR_TECHNCL_CONTACT_NM		
L1000A_S06_PER04_PAYR_CONTACT_COMNCTN_NR_OVL_TELPHN		
L1000B_S01_N102_PAY_NM		
L1000B_S01_N104_ID_CD_OVL_FED_TAX_ID_NR		
L1000B_S02_N301_ADRS_LIN		

L1000B_S03_N401_CITY_NM		
L1000B_S03_N402_STAT_CD		
L1000B_S03_N403_PAY_POSTL_ZON_ZIP_CD		
L2000_S01_LX01_ASGND_NR		
L2100_S01_CLP01_PATNT_CONTRL_NR		
L2100_S01_CLP02_STATS_CD		
L2100_S01_CLP03_TOTL_CLM_CHG_AMT		
L2100_S01_CLP04_PMT_AMT		
L2100_S01_CLP06_CLM_FILNG_IND_CD		
L2100_S01_CLP07_PAYR_CLM_CONTRL_NR		
L2100_S03_NM103_PATNT_LNAME		
L2100_S03_NM104_PATNT_FNAME		
L2100_S03_NM109_PATNT_ID_OVL_MEM_ID_NR		
L2100_S14_DTM02_CLM_DT_OVL_CLM_STATMNT_PERD_START		
L2100_S14_DTM02_I02_CLM_DT_OVL_CLM_STATMNT_PERD_END		
L2100_S18_AMT02_CLM_SUPP_NFO_AMT_OVL_COVG_AMT		
L2110_S01_SVC01_02_ADJDCTD_PROC_CD_OVL_HCPCS_CD		
L2110_S01_SVC02_LIN_ITM_CHG_AMT		
L2110_S01_SVC03_LIN_ITM_PROV_PMT_AMT		
L2110_S01_SVC05_UNTS_SVC_PD_CT		
L2110_S02_DTM02_SVC_DT_OVL_SVC		
L2110_S03_CAS01_CLM_ADJ_GRP_CD		
L2110_S03_CAS02_ADJ_RSN_CD		
L2110_S03_CAS03_ADJ_AMT		

BIN_837D Table

This BIN table holds mappings matching that of a minimal 5010 837 Dental A2 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
V2_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		

GSHDR_S01_GS06_GCN		
GSHDR_S01_GS08_VERS_ID_CD		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S02_BHT01_STRUCTR_CD		
STHDR_S02_BHT02_TS_PURPS_CD		
STHDR_S02_BHT03_ORGNTR_APLCTN_TRANSCTN_ID		
STHDR_S02_BHT04_TS_CREATN_DT		
STHDR_S02_BHT05_TS_CREATN_TIM		
STHDR_S02_BHT06_CLM_ENCNTR_ID		
STHDR_S06_SE02_TCN		
L1000A_S01_NM103_SUBMTR_LAST_ORG_NM_OVL_NONPRSN_ENTY		
L1000A_S01_NM109_SUBMTR_ID_OVL_ETN_NR		
L1000A_S02_PER02_CONTCT_NM		
L1000A_S02_PER04_COMM_NR_OVL_TELPHN		
L1000B_S01_NM103_RECVR_NM_OVL_NONPRSN_ENTY		
L1000B_S01_NM109_PRIMARY_ID_OVL_ETN_NR		
L2000A_S01_HL01_HIERCHCL_ID_NR		
L2010AA_S01_NM103_LAST_ORGL_NM_OVL_NONPRSN_ENTY		
L2010AA_S02_N301_BILNG_PROV_ADRS_LIN		
L2010AA_S03_N401_BILNG_PROV_CITY_NM		
L2010AA_S03_N402_STAT_PROVNC_CD		
L2010AA_S03_N403_POSTL_ZON_ZIP_CD		

L2010AA_S04_REF02_ID_NR_OVL_EMPLYR_ID_NR		
L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000B_S01_HL04_HIERCHCL_CHILD_CD		
L2000B_S02_SBR01_PAYR_RESP_SEQNC_NR_CD		
L2000B_S02_SBR02_INDVDL_REL_CD		
L2000B_S02_SBR09_CLM_FILNG_IND_CD		
L2010BA_S01_NM103_SUB_LNAME_OVL_PERSN		
L2010BA_S01_NM104_SUB_FNAME		
L2010BA_S01_NM109_PRIMARY_ID_OVL_MEM_ID_NR		
L2010BA_S02_N301_ADRS_LIN		
L2010BA_S03_N401_CITY_NM		
L2010BA_S03_N402_STAT_CD		
L2010BA_S03_N403_SUB_POSTL_ZON_ZIP_CD		
L2010BA_S04_DMG02_BIRTH_DT_OVL_DT_CCYYMMDD		
L2010BA_S04_DMG03_GENDR_CD		
L2010BB_S01_NM103_PAYR_NM_OVL_NONPRSN_ENTY		
L2010BB_S01_NM109_PAYR_ID_OVL_PAYR_ID		
L2300_S01_CLM01_PATNT_CONTRL_NR		
L2300_S01_CLM02_TOTL_CLM_CHG_AMT		
L2300_S01_CLM05_01_PLAC_SVC_CD		
L2300_S01_CLM05_03_CLM_FREQNCY_CD_OVL_PLAC_SVC_CODS_PROF_DENTL		
L2300_S01_CLM06_PROV_SUPLR_SIGNTR_IND		

L2300_S01_CLM07_ASGNMNT_PLAN_PARTCPTN_CD		
L2300_S01_CLM08_BENFTS_ASGNMNT_CERT_IND		
L2300_S01_CLM09_RELS_NFO_CD		
L2300_S04_DTP03_SVC_DT_OVL_SVC_RD8		
L2400_S01_LX01_ASGND_NR		
L2400_S02_SV301_02_PROC_CD_OVL_AMRCN_DENTL_ASCTN_CD		
L2400_S02_SV302_LIN_ITM_CHG_AMT		

BIN_837I Table

This BIN table holds mappings matching that of a minimal 5010 837 Institutional A2 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
W2_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		
GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		

GSHDR_S01_GS08_VERS_ID_CD		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S02_BHT01_STRUCTR_CD		
STHDR_S02_BHT02_TS_PURPS_CD		
STHDR_S02_BHT03_ORGNTR_APLCTN_TRANSCTN_ID		
STHDR_S02_BHT04_TS_CREATN_DT		
STHDR_S02_BHT05_TS_CREATN_TIM		
STHDR_S02_BHT06_CLM_ID		
STHDR_S06_SE02_TCN		
L1000A_S01_NM103_SUBMTR_LAST_ORG_NM_OVL_NONPRSN_ENTY		
L1000A_S01_NM109_SUBMTR_ID_OVL_ETN_NR		
L1000A_S02_PER02_CONTACT_NM		
L1000A_S02_PER04_COMM_NR_OVL_TELPHN		
L1000B_S01_NM103_RECVR_NM_OVL_NONPRSN_ENTY		
L1000B_S01_NM109_PRIMARY_ID_OVL_ETN_NR		
L2000A_S01_HL01_HIERCHCL_ID_NR		
L2010AA_S01_NM103_BILNG_PROV_ORGL_NM_OVL_NONPRSN_ENTY		
L2010AA_S02_N301_BILNG_PROV_ADRS_LIN		
L2010AA_S03_N401_BILNG_PROV_CITY_NM		
L2010AA_S03_N402_STAT_PROVNC_CD		
L2010AA_S03_N403_POSTL_ZON_ZIP_CD		
L2010AA_S04_REF02_ID_NR_OVL_EMPLYR_ID_NR		

L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000B_S01_HL04_HIERCHCL_CHILD_CD		
L2000B_S02_SBR01_PAYR_RESP_SEQNC_NR_CD		
L2000B_S02_SBR02_INDVDL_REL_CD		
L2000B_S02_SBR09_CLM_FILNG_IND_CD		
L2010BA_S01_NM103_SUB_LNAME_OVL_PERSN		
L2010BA_S01_NM104_SUB_FNAME		
L2010BA_S01_NM109_PRIMRY_ID_OVL_MEM_ID_NR		
L2010BA_S02_N301_ADRS_LIN		
L2010BA_S03_N401_CITY_NM		
L2010BA_S03_N402_STAT_CD		
L2010BA_S03_N403_SUB_POSTL_ZON_ZIP_CD		
L2010BA_S04_DMG02_BIRTH_DT_OVL_DT_CCYYMMDD		
L2010BA_S04_DMG03_GENDR_CD		
L2010BB_S01_NM103_PAYR_NM_OVL_NONPRSN_ENTY		
L2010BB_S01_NM109_PAYR_ID_OVL_PAYR_ID		
L2010BB_S02_N301_ADRS_LIN		
L2010BB_S03_N401_CITY_NM		
L2010BB_S03_N402_STAT_CD		
L2010BB_S03_N403_PAYR_POSTL_ZON_ZIP_CD		
L2300_S01_CLM01_PATNT_CONTRL_NR		
L2300_S01_CLM02_TOTL_CLM_CHG_AMT		

L2300_S01_CLM05_01_FACLT_YTYPE_CD		
L2300_S01_CLM05_03_CLM_FREQNCY_CD_OVL_UBC_FORM_BIL_TYPE		
L2300_S01_CLM07_ASGNMNT_PLAN_PARTCPTN_CD		
L2300_S01_CLM08_BENFTS_ASGNMNT_CERT_IND		
L2300_S01_CLM09_RELS_NFO_CD		
L2300_S03_DTP03_STATMNT_FROM_TO_DT_OVL_STATMNT_RD8		
L2300_S06_CL101_ADMSN_TYPE_CD		
L2300_S06_CL102_ADMSN_SOURC_CD		
L2300_S06_CL103_PATNT_STATS_CD		
L2300_S26_HI01_02_PRINCPL_DIAG_CD_OVL_ICD9CM_PRINCPL_DIAG		
L2300_S31_HI01_02_OTHR_DIAG_OVL_ICD9CM_DIAG		
L2310A_S01_NM103_PROV_LNAME_OVL_PERSN		
L2310A_S01_NM104_PROV_FNAME		
L2310A_S03_REF02_ATNDNG_PROV_2ND_ID_OVL_PROV_COMRCL_NR		
L2400_S01_LX01_ASGND_NR		
L2400_S02_SV201_SVC_LIN_REVN_CD		
L2400_S02_SV202_02_PROC_CD_OVL_HCPCS_CD		
L2400_S02_SV203_LIN_ITM_CHG_AMT		
L2400_S02_SV205_UNT_CT_OVL_UNT		
L2400_S04_DTP03_SVC_DT_OVL_SVC_D8		

BIN_837P Table

This BIN table holds mappings matching that of a minimal 5010 837 Professional A2 EDI file. For additional possible mappings, see the CGIF reference HTML file in the C:\chiapas\docs\CGIF folder.

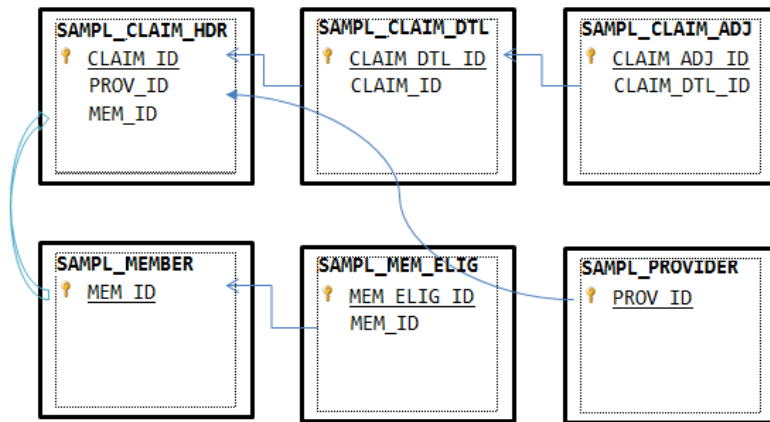
Column Name	Data Type	Purpose
BIN_PK	INT, PK	Primary Key (Identity)
BIN_ID	INT	Foreign key to the BIN_LOG table
X1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO		
OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO		
OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID		
OEISA_S01_ISA09_DT		
OEISA_S01_ISA10_TM		
OEISA_S01_ISA11_REPTN_SEPRTR		
OEISA_S01_ISA12_VERSN_NR		
OEISA_S01_ISA13_ICN		
OEISA_S01_ISA14_ACK_REQ		
OEISA_S01_ISA15_USG_IND		
OEISA_S01_ISA16_SUBELE_SEP		
OEISA_S03_IEA01_GS_CT		
OEISA_S03_IEA02_ICN		
GSHDR_S01_GS02_APP_SENDR_CD		
GSHDR_S01_GS03_APP_RECVR_CD		

GSHDR_S01_GS04_DT		
GSHDR_S01_GS05_TM		
GSHDR_S01_GS06_GCN		
GSHDR_S01_GS08_VERS_ID_CD		
GSHDR_S03_GE01_TS_CT		
GSHDR_S03_GE02_GCN		
STHDR_S01_ST02_CONTRL_NR		
STHDR_S02_BHT01_STRUCTR_CD		
STHDR_S02_BHT02_TS_PURPS_CD		
STHDR_S02_BHT03_ORGNTR_APLCTN_TRANSCTN_ID		
STHDR_S02_BHT04_TS_CREATN_DT		
STHDR_S02_BHT05_TS_CREATN_TIM		
STHDR_S02_BHT06_CLM_ENCNTR_ID		
STHDR_S06_SE01_SEG_CT		
STHDR_S06_SE02_TCN		
L1000A_S01_NM103_SUBMTR_LAST_ORG_NM_OVL_NONPRSN_ENTY		
L1000A_S01_NM109_SUBMTR_ID_OVL_ETN_NR		
L1000A_S02_PER02_CONTCT_NM		
L1000A_S02_PER04_COMM_NR_OVL_TELPHN		
L1000B_S01_NM103_RECVR_NM_OVL_NONPRSN_ENTY		
L1000B_S01_NM109_PRIMARY_ID_OVL_ETN_NR		
L2000A_S01_HL01_HIERCHCL_ID_NR		
L2010AA_S01_NM103_LAST_ORGL_NM_OVL_NONPRSN_ENTY		

L2010AA_S02_N301_BILNG_PROV_ADRS_LIN		
L2010AA_S03_N401_BILNG_PROV_CITY_NM		
L2010AA_S03_N402_STAT_PROVNC_CD		
L2010AA_S03_N403_POSTL_ZON_ZIP_CD		
L2010AA_S04_REF02_ID_NR_OVL_EMPLYR_ID_NR		
L2000B_S01_HL01_HIERCHCL_ID_NR		
L2000B_S01_HL02_HIERCHCL_PARNT_ID_NR		
L2000B_S01_HL04_HIERCHCL_CHILD_CD		
L2000B_S02_SBR01_PAYR_RESP_SEQNC_NR_CD		
L2000B_S02_SBR02_INDVDL_REL_CD		
L2000B_S02_SBR09_CLM_FILNG_IND_CD		
L2010BA_S01_NM103_SUB_LNAME_OVL_PERSN		
L2010BA_S01_NM104_SUB_FNAME		
L2010BA_S01_NM109_PRIMARY_ID_OVL_MEM_ID_NR		
L2010BA_S02_N301_ADRS_LIN		
L2010BA_S03_N401_CITY_NM		
L2010BA_S03_N402_STAT_CD		
L2010BA_S03_N403_SUB_POSTL_ZON_ZIP_CD		
L2010BA_S04_DM02_BIRTH_DT_OVL_DT_CCYYMMDD		
L2010BA_S04_DM03_GENDR_CD		
L2010BB_S01_NM103_PAYR_NM_OVL_NONPRSN_ENTY		
L2010BB_S01_NM109_PAYR_ID_OVL_PAYR_ID		
L2010BB_S02_N301_ADRS_LIN		

L2010BB_S03_N401_CITY_NM		
L2010BB_S03_N402_PAYR_STAT_PROVNC_CD		
L2010BB_S03_N403_PAYR_POSTL_ZON_ZIP_CD		
L2300_S01_CLM01_PATNT_CONTRL_NR		
L2300_S01_CLM02_TOTL_CLM_CHG_AMT		
L2300_S01_CLM05_01_PLAC_SVC_CD		
L2300_S01_CLM05_03_CLM_FREQNCY_CD_OVL_PLAC_SVC_CODS_PROF_DENTL		
L2300_S01_CLM06_PROV_SUPLR_SIGNTR_IND		
L2300_S01_CLM07_ASGNMNT_PLAN_PARTCPTN_CD		
L2300_S01_CLM08_BENFTS_ASGNMNT_CERT_IND		
L2300_S01_CLM09_RELS_NFO_CD		
L2300_S43_HI01_02_DIAG_CD_OVL_ICD9CM_PRINCPL_DIAG		
L2300_S43_HI02_02_DIAG_CD_OVL_ICD9CM_DIAG		
L2310B_S01_NM103_LAST_ORG_NM_OVL_NONPRSN_ENTY		
L2310B_S03_REF02_RENDRNG_PROV_2ND_ID_OVL_PROV_COMRCL_NR		
L2400_S01_LX01_ASGND_NR		
L2400_S02_SV101_02_PROC_CD_OVL_HCPCS_CD		
L2400_S02_SV102_LIN_ITM_CHG_AMT		
L2400_S02_SV104_SVC_UNT_CT_OVL_UNT		
L2400_S02_SV107_01_DIAG_CD_POINTR		
L2400_S11_DTP03_SVC_DT_OVL_SVC_D8		

SAMPLE OBJECT SCHEMA



In order to demonstrate the process of encoding different specifications, Chiapas EDI Enterprise stores mock member, claims and provider data in sample tables included with the distribution database. These tables represent mock claims, services, adjustments, members, enrollments and providers that can be used in different ways to generate sample files.

This data supports the vw_Export ETL views included with the distribution database so that there is a built-in pathway to creating any of the 13 EDI transactions quickly. This data combined with the ETL views provide a fixed starting point for a developers wishing to create outbound transactions from their own enterprise data.

SAMPL CLAIM HDR Table

This table stores mock Claim header-level data.

Column Name	Data Type	Purpose
CLAIM_ID	INT, PK	Unique Claim Identifier
PROV_ID	INT	Foreign Key to Provider table
MEM_ID	INT	Foreign Key to Member table
PATNT_ACCT_NR	VARCHAR(50)	Patient Account Number
FAC_TYP_CD	VARCHAR(2)	Facility Type Code
PRI_DIAG_CD	VARCHAR(10)	Primary Diagnosis Code
SEC_DIAG_CD	VARCHAR(10)	Secondary Diagnosis Code
STATUS	VARCHAR(10)	Claim Adjudication Status
ADJUDCTN_DT	DATE	Claim Adjudication Date

SAMPL_CLAIM_DTL Table

This table stores mock Claim Detail data, with a foreign key to the SAMPL_CLAIM_HDR table.

Column Name	Data Type	Purpose
CLAIM_DTL_ID	INT, PK	Unique Claim Detail Line Identifier
CLAIM_ID	INT	Foreign Key reference to the Claim Header table
CHG_AMT	MONEY	Claim Line Charge Amount
COPAY_AMT	MONEY	Claim Line Patient Copay Amount
PAID_AMT	MONEY	Claim Line Paid Amount
FROM_DT	DATE	Claim Line Service From Date
TO_DT	DATE	Claim Line Service To Date
SVC_CD	VARCHAR(10)	Claim Line Procedure Code
UNIT_CT	INT	Claim Line Unit Count

SAMPL_CLAIM_ADJ Table

This table stores sample claim service line adjustment information, with a foreign key link to the Claim Detail table.

Column Name	Data Type	Purpose
CLAIM_ADJ_ID	INT, PK	Unique Claim Detail Adjustment Identifier
CLAIM_DTL_ID	INT	Foreign Key reference to the Claim Detail table
ADJ_REASN	VARCHAR(2)	Adjustment Reason Code
ADJ_AMT	MONEY	Adjustment Amount
ADJ_SEQ	INT	Adjustment Sequence

SAMPL_MEMBER Table

This table stores mock members. The last names are all set to 'food names' so that files generated with this data will not be mistaken for real Protected Healthcare Information.

Column Name	Data Type	Purpose
MEM_ID	INT, PK	Unique Member Adjustment Identifier
FIRST_NM	VARCHAR(50)	First Name
LAST_NM	VARCHAR(50)	Last Name
SSN_ID	VARCHAR(20)	Social Security Number
BIRTH_DT	DATE	Birth Date
GENDER	VARCHAR(1)	Gender
ADD1	VARCHAR(75)	Address Line 1
ADD2	VARCHAR(75)	Address Line 2
CITY	VARCHAR(50)	City
STATE	VARCHAR(2)	State
ZIP_CD	VARCHAR(20)	Zip Code
PHONE_NR	VARCHAR(20)	Phone Number

SAMPL_MEM_ELIG Table

This table stores sample member enrollment span information.

Column Name	Data Type	Purpose
MEM_ELIG_ID	INT, PK	Unique Member Eligibility Identifier
MEM_ID	INT	Foreign Key to the Member table
STATUS	VARCHAR(10)	Eligibility status
FROM_DT	DATE	Eligibility From Date
TO_DT	DATE	Eligibility To Date

SAMPL_PROVIDER Table

The table stores a number of mock providers.

Column Name	Data Type	Purpose
PROV_ID	INT, PK	Unique Provider Identifier
TAX_ID	VARCHAR(15)	Federal Tax ID
NPI_ID	VARCHAR(12)	National Provider Identifier
COMPANY_NM	VARCHAR(50)	Company Name
FIRST_NM	VARCHAR(20)	First Name
LAST_NM	VARCHAR(20)	Last Name
ADD1	VARCHAR(75)	Address Line 1
ADD2	VARCHAR(75)	Address Line 2
CITY	VARCHAR(50)	City
STATE	VARCHAR(2)	State
ZIP_CD	VARCHAR(20)	ZIP Code
CONTACT_NM	VARCHAR(50)	Contact Name
WORK_PH	VARCHAR(50)	Work Phone
EMAIL	VARCHAR(50)	Email

SEED TRANSACTIONS

Chiapas EDI Enterprise supports thirteen different transaction types, and provides a number of different resources developers can use for development purposes:

SEED FILES

Located in the C:\chiapas\samples directory, these files represent fully tested, HIPAA compliant transactions as generated from the distribution environment. For developers needing to generate outbound transactions, these can be used as a starting point that can be tailored to specific business purposes.

ETL Export Views

The seed files are generated by the vw_Export_* views present within the distribution database. They generate CGIF Flat recordsets. To see the files being generated, open up Chiapas Studio and follow these steps:

1. Make sure the Flat / DB Table tab is opened selected and the Table to Flat radio button is selected.
2. Ensure the Data Source is correct to the chiapasData database.
3. In the Table window, enter:

```
EXEC USP_ORDEREDI 'vw_Export_834'
```

4. Press Execute in the DB Table sub-tab.
5. Under the HKey pulldown menu, select 'Flat to HKey'
6. Press the Encode button.
7. Select the SegPool Data I/O tab, select the SegPool to File radio button.
8. Enter in a new filename in the SegPool File box.
9. Press Execute.

These are the steps necessary to regenerate any of the seed files from the sample data.

The views themselves are standard T/SQL code written to create the standard EDI transactions from the SAMPL mock healthcare data tables. They can be considered the minimal necessary code needed to create a HIPAA compliant transaction; by translating the view away from the SAMPL data tables and to a backend enterprise system, new outbound transactions can be made quickly.

All of the ETL extracts have the following in common:

ISA/GS/ST envelope section:

The first section of all of the ETL views is dedicated to providing valid data for the envelopes and correct mappings for the transactions they are in. One important point to note is that Chiapas EDI Enterprise is capable of generating certain mappings, like timestamps and segment counts, automatically. If the mapping is not supplied to the encoder and the encoder has a default value to place there, then that mapping will be automatically handled by the encoder engine.

Transaction Body:

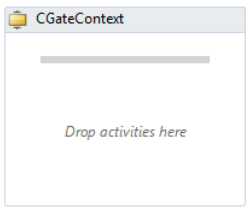
The second section is dedicated to all of the specific mapped elements for a minimal implementation of each transaction set. Here, the member, provider and claim tables are used as a data source for dummy information.

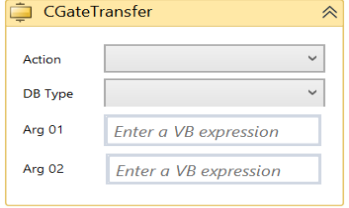
Two views deserve special mention:

vw_Export_835 generates a valid 835 remittance file with the use of a 'base' view that organizes claims, services and adjustments. This base view is referenced several times, one to generate aggregate sums and the other as the row data source for the extract. The actual vw_Export_835 ETL view then uses this information to project valid 835 elements, using the built-in function fx_HIPAA_NUMBER to convert database monetary numbers into HIPAA compliant number formatting.

vw_Sample_270_XML is a demonstration of using SQL Server XML generation syntax to create a CGIFXML2 view of the 270 sample data. This XML can be saved to a file and imported to the HKey via Chiapas Studio or imported via the XML Decode workflow.

WORKFLOW ACTIVITIES

	CGateContext	
	Parameters	None
Error Messages	None	
Purpose: <p>This activity is the main container for all other Chiapas EDI activities. It is defined as a special ‘Sequence’ activity – to add more Activities inside, they need to be dragged over the grey rectangles inside. CGateContext activities cannot be nested – you cannot place one CGateContext inside of another. Furthermore, all of the other defined Chiapas EDI activities must be placed within a CGateContext container – placing one outside of a CGateContext will result in an error.</p> <p>The CGateContext stores the ‘State’ of the Chiapas engine, and holds a single instance of all registers. A single instance of the CGateContext activity can be used for both a single Encode and Decode operation. When the workflow is initially run, it creates a new state object and opens a connection to the chiapasData database named ‘TriggerDB’. The Worker Process passes an single argument, the EVENT_ID, which is available via the CGateQuery activity. After the workflow completes execution, the CGateContext activity sends any errors or messages that were generated to the database.</p> <p>Note: The State object is engineered to handle a single transaction or file at a time. It is capable of running both encode and decode operations, but should not be used with looping structure activities to process more than one file at a time. Chiapas EDI Enterprise is capable of processing multiple files via concurrent workflows and the automation system.</p>		

	<div>CGateTransfer</div> <table> <tr> <td>Parameters</td><td> DataArg01 (string) DataArg02 (string) DBType (string) TransferType (string) </td></tr> </table>	Parameters	DataArg01 (string) DataArg02 (string) DBType (string) TransferType (string)
Parameters	DataArg01 (string) DataArg02 (string) DBType (string) TransferType (string)		
Error Messages	<p>ErrLog ErrLog objects can only output to CSV or Database, data cannot be bulk loaded into an ErrLog object.</p> <p>The ErrLog CNN commands require 2 arguments for a Database endpoint or 1 argument for a CSV file endpoint</p> <p>ErrLog objects are limited to FlatCSV or Flat Database type data endpoints.</p> <p>Error during storing of ErrLog to Database</p> <p>Error occurred while writing the CSV file</p> <p>Flat Flat objects are limited to Table, CSV or Excel endpoints</p> <p>The Flat commands require 2 arguments for a Database endpoint or 1 argument for a CSV file endpoint</p> <p>Error during storing of Flat to Database</p> <p>Error during loading of Flat from Database</p> <p>The destination file already exists.</p> <p>The source file does not exist.</p> <p>Error during storing of Flat to CSV</p> <p>Error during loading Flat from CSV</p> <p>Error during storing of Flat to Excel</p> <p>Error during loading Flat from Excel</p> <p>HKey HKey objects are limited to HKey Tables & XML endpoints</p>		

	<p>The HKey connector command requires one parameter for an XML File endpoint, two arguments for HKey DB Tables or three arguments for an HKey DB endpoint</p> <p>There was a problem storing the HKey to an XML file</p> <p>There was a problem loading the HKey from an XML file</p> <p>There was a problem loading the HKey from HKey DB Tables</p> <p>There was a problem loading the HKey from the XML DB table</p> <p>There was a problem saving the HKey to HKey DB Tables</p> <p>There was a problem saving the HKey to an XML DB Table</p> <p>SegPool SegPool objects are limited to SegPool / SegPoolAck file endpoints</p> <p>SegPoolAck operations need a concurrently loaded SegPool object on which to function - spawning an empty SegPool.</p> <p>SegPool objects require 1 argument</p> <p>Specification cannot be determined from the file - transfer failed.</p> <p>SegPool Load error: Invalid ISA/GS headers; specification cannot be determined.</p> <p>There was a problem loading the SegPool / SegPoolAck from the file endpoint</p> <p>There was a problem saving the SegPool / SegPoolAck to a file endpoint</p>
<p>Purpose:</p> <p>This single activity provides an interface to accessing nearly all of the major I/O functions of the platform. Most of the I/O operations from the Data I/O tab of ChiapasStudio can be performed within this single activity. This activity has a designer interface that switches according to context to make it easier to use the various operations.</p> <p>ErrLog Register</p> <p>This activity will send all errors and messages generated so far in this workflow to either a database table or CSV file. All errors are automatically sent to the MSGLOG table upon completion of the workflow; therefore, this activity is useful when running workflows directly from the Workflow Studio or when debugging workflows that generate critical errors that disrupt the normal logging system.</p>	

ErrLog > DB Table

DB Type: Database Type

Data Source: Valid Connection String

Data Table: This is the name of a database table where Chiapas EDI Enterprise will send the ErrLog register. It must follow the schema of the MSGLOG table within the chiapasData database.

ErrLog > CSV File

File Path: Name of the file to create.

Flat Register

These actions load or save the Flat register.

Flat > DB Table

DB Type: Database Type

Data Source: Valid Connection String

Data Table: Destination table to receive Flat register

This action exports a loaded Flat register to a database table. If the database type selected is SQLite or SQL Server, the destination table will be created if it doesn't exist. Then, all rows of the Flat object will be inserted. If the table previously exists and there is a mismatch between the schemas for the Flat register and the data table, then no rows will be inserted and an error will be logged.

Flat < DB Table

DB Type: Database Type

Data Source: Valid Connection String

Data Table: SQL Tableset source

This action loads the Flat register from a CGIF Flat formatted database table source. It will accept a database table, a SELECT SQL statement, or a stored procedure that is prefixed by EXEC. Some examples:

```
vw_Export_837I
SELECT * FROM vw_Export_834 ORDER BY L2000_S04_REF02_SUPP_ID_OVL_CLIENT_REPRTNG_CATGRY
EXEC USP_ORDEREDI 'vw_Export_834'
```

Note: Chiapas EDI Enterprise relies on data being sorted prior to it being supplied to the encoder engine. In the above examples, the first View cannot guarantee properly sorted rows so that claims are aggregated together correctly. The second example does some simple sorting but this may not be sufficient in a production environment. The third example uses a built-in stored procedure that automatically sorts the CGIF Flat data in the correct order.

Flat > CSV File

File Path: CGIF Flat CSV file to create

Flat < CSV File

File Path: CGIF Flat CSV file to load

These actions export or import the Flat register in the form of a CSV file. All values will be surrounded by double quotes and comma separated, and will have a header row.

Flat > Excel File

File Path: CGIF Flat Excel file to create (.xls and .xlsx file types supported)

Flat < Excel File

File Path: CGIF Flat Excel file to load (.xls and .xlsx file types supported)

These actions export or import the Flat register in the form of an Excel spreadsheet. The filename extension used will determine if the file is an old-style Excel '97 format file or the newer Excel 2003 file format. Note that the older Excel format is very restrained in its' ability to store larger files.

HKey > XML File

File Path: CGIFXML2 file to create

HKey < XML File

File Path: CGIFXML2 file to load

This loads and saves the HKey register in CGIFXML2 format.

HKey > XML File (Legacy)

File Path: CGIFXML file to create

HKey < XML File (Legacy)

File Path: CGIFXML file to load

This is an older, legacy XML type that is being phased out of the product in favor of the more flexible CGIFXML2 data type.

HKey > DB Tables

DB Type: SQL Server

Data Source: Valid SQL Connection String

Prefix: Character prefix for the HKey storage tables

This command creates a set of discrete database tables where each loop is stored as a separate table. Numeric fields (KEY_ID and PAR_KEY_ID) link child loop tables to their parents, with the root parent ISA table having no PAR_KEY_ID set. The character(s) indicated in the 'Prefix' will be inserted before each loop table name.

HKey > DB View

DB Type: SQL Server

Data Source: Valid SQL Connection String

Prefix: Character prefix for the HKey storage tables

This command can only be run immediately following another CGateTransfer of 'HKey > DB Tables' (above). It will create a SQL View called 'VW_HKEY_<Prefix>' on the database indicated within the connection string. The resulting dataset of the SQL View is the 'Flat' projection of the hierarchical data, the same as pressing the 'HKey < Flat' button on Chiapas Studio.

HKey < DB Tables

DB Type: SQL Server

Data Source: Valid SQL Connection String

Prefix: Character prefix for the HKey storage tables

This command loads the HKey with hierarchical data as if it was generated in the 'HKey > DB Tables' function above. The specification will be obtained via the first mapping in the ISA table, and then each of the loops possible for that specification will be queried on the target SQL Server and loaded to the internal HKey register.

SegPool > File

File Path: EDI file to create

SegPool < File

File Path: EDI file to load

This action imports and exports the SegPool register. Note that any formatting options you'd like to set on a generated EDI file need to be set *prior* to the Encode activity.

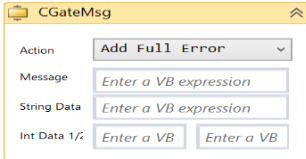
SegPoolAck > File

File Path: 999 File to Create

SegPoolAck < File

File Path: 999 File to Load

This activity handles I/O for the SegPoolAck register, which is an internal representation of a 999 file. See the CGateAckParse and CGateAckGenerate activities for more information.

	<h3>CGateMsg</h3>	
ErrorMessages	None	
<p>Purpose:</p> <p>This activity controls the ErrLog register. It can be used to clear the message log or create custom user messages. When a workflow completes, all messages in the ErrLog register are automatically recorded to the chiapasData MSGLOG table.</p> <p>Actions:</p> <p><u>Add Simple Message</u> This adds a message to the message log</p> <p><u>Add Full Message</u> This action adds a complex message to the log, including string data and numeric data values.</p> <p><u>Add Simple Error</u></p>		

Adds a single message to the log and flags as an error

Add Full Error

Adds a complex error and flags it as an error.

Clear

Clears the message log

	<div>CGateAckParse</div> <div>ParametersNone</div>
Error Messages	<div>SegPoolAck Register is not loaded</div> <div>Unrecognized Segment ID</div> <div>Unexpected Segment</div> <div>Required Segment Missing</div> <div>Loop Occurs Over Maximum Times</div> <div>Segment Exceeds Maximum Use</div> <div>Segment not in Defined Transaction Set</div> <div>Segment Not in Proper Sequence</div> <div>Segment Has Data Element Errors</div> <div>Implementation NOT USED Segment Present</div> <div>Implementation Dependent Segment Missing</div> <div>Implementation Loop Occurs Under Minimum Times</div> <div>Implementation Segment below Minimum Use</div> <div>Implementation Dependent NOT USED Segment Present</div> <div>Mandatory Data Element missing</div> <div>Conditionally Required Data Element Missing</div> <div>Too Many Data Elements</div> <div>Data Element is Too Short</div> <div>Data Element is Too Long</div> <div>Invalid Character in Data Element</div> <div>Invalid Code Value</div> <div>Invalid Date</div> <div>Invalid Time</div> <div>Exclusion Condition Violated</div> <div>Too Many Repetitions</div> <div>Too Many Components</div> <div>Implemented NOT USED Data Element Present</div> <div>Implementation Too Few Repetitions</div> <div>Implementation Pattern Match Failure</div> <div>Implementation Dependent NOT USED Data Element Present</div> <div>Code Value Not Used in Implementation</div> <div>Implementation Dependent Data Element Missing</div>
<div>Purpose:</div> <div> <p>This activity translates the SegPoolAck register into a series of human-readable error messages associated with segments in the SegPool and writes them to the log registers. These messages can later be reviewed to determine the cause of integrity issues.</p> <p><u>Loading 999s:</u></p> <ol style="list-style-type: none"> 1. SegPool is loaded earlier in workflow 2. SegPoolAck < File operation on this activity loads the 999 <p>CGateAckParse will parse out the messages where they will be placed in the MSGLOG table when the workflow completes.</p> </div>	

	CGateAckGenerate	
	Parameters	None
Error Messages	None	
Purpose:		
<p>This activity loads the SegPoolAck register after a CGateXltDecode activity decodes a file. Chiapas EDI Enterprise’s 999 implementation is very simple:</p>		
<ul style="list-style-type: none">- All of the header information present within the SegPool is reversed (receiver becomes sender)- If the file was able to be completely parsed without critical parsing errors, a ‘Success’ acknowledgment is generated for each of the transactions encountered in the SegPool.- If the file encountered a critical decoding error, a rejection indicating <i>only</i> the errored transaction and errored segment is generated.		
Creating new 999s:		
<ol style="list-style-type: none">1. SegPool is loaded earlier in workflow2. Decode operation is executed3. CGateAckGenerate activity loads the SegPoolAck register depending on the outcome of the Decode operation4. CGateTransfer sends the SegPoolAck register to an EDI file.		

	CGateAckTA1Generate	
	Parameters	None
Error Messages	None	
Purpose: This activity loads the SegPoolAck register after a CGateXltDecode activity decodes a file. Chiapas EDI Enterprise'sTA1 implementation is very simple: <ul style="list-style-type: none">- Every ISA/IEA loop is acknowledged with a single TA1 segment. It will list 'A' if there were no critical errors during decode, or 'R' if there were. Creating new TA1s: <ol style="list-style-type: none">5. SegPool is loaded earlier in workflow6. Decode operation is executed7. CGateAckTA1Generate activity loads the SegPoolAck register depending on the outcome of the Decode operation8. CGateTransfer sends the SegPoolAck register to an EDI file.		

	CGateSeg	
	Parameters	bol80Col (true / false) bolCR (true / false) bolDelPHI (true / false) bolLF (true / false) EleSep (string) SegSep (string) SubEleSep (string)
Error Messages	None	
Purpose: This activity will set a number of options on the SegPool register that will affect any new SegPool registers created by the CGateXltEncode activity. These options will set the Segment Separator, Element Separator, Sub-Element Separator, and on the right, Carriage Return at the end of a segment, Line Feed at the end of a segment, 80 columns split, and whether to scrub Protected Healthcare Information. This activity needs to be present <i>prior</i> to encoding of the file for these formatting options to be applied on the final outgoing file.		

	CGateHKeyFromFlat	
	Parameters	None
Error Messages	Exception while transferring Flat to HKey	
	Error: Invalid Mapping	
Purpose: This activity will translate the Flat register to the HKey register, transforming the two-dimensional column/row layout of the Flat register into the hierarchical format of the HKey register. This activity is necessary any time the Flat register is loaded so that it can be transformed into an HKey and finally into an EDI file.		

	CGateHKeyToFlat	
	Parameters	None
Error Messages	Exception while transferring HKey to Flat	
Purpose:		

The counterpart to the CGateHKeyFromFlat activity, this will transform the HKey to the two dimensional Flat register.

	CGateHKeyFromXMLDB	
	Parameters	SrcCnn (string) SrcTable (string) SrcXMLID (string)
Error Messages	CGateHKeyFromXMLDB Exception: There was a problem loading the HKey from the XML DB table	
Purpose: This activity enables loading the HKey register directly from a table that follows the BIN_XML schema. The CGIFXML2 XML data is taken from the XML_STOR column from the row that matches the XML_ID given in the SrcXMLID argument.		

	CGateHKeyToXMLDB	
	Parameters	DestCnn (string) DestTable (string)
Error Messages	CGateHKeyToXMLDB Exception: There was a problem loading the HKey from the XML DB table	
Purpose: This activity enables loading the HKey register directly from a table that follows the BIN_XML schema. The CGIFXML2 XML data is taken from the XML_STOR column from the row that matches the XML_ID given in the SrcXMLID argument. Both the EVENT_ID and the XML_ID will be populated automatically.		

	CGateHKeyToXElement	
	Parameters	HKeyXML (XElement)
Error Messages	CGateHKeyToXElement Exception:	
Purpose: This activity loads a supplied XElement variable (referenced in the System.Xml.Linq library) with an XML document representing the currently loaded HKey register.		

	CGateXltEncode	
	Parameters	None
Error Messages	Invalid CGIF Mapping The Encoder was unable to initialize for the encoding process. This is can be caused by mandatory loops that do not have any supplied data mappings, for example, HL loops. If this is the case, make sure HL04 (Child Code) is supplied for HL loops.	
Purpose: This activity transforms the loaded HKey register to the SegPool register, effectively encoding a new EDI file from the supplied data. It will attempt to encode all required elements and segments – if a required element is not supplied from the HKey register, it will default that element with whitespace. The following elements may be defaulted by the Encoder engine:		
Defaulted Element	Defaulted Value	
ISA01	00	
ISA02	Whitespace	
ISA03	00	
ISA04	Whitespace	
ISA05	Sender Qual - Value set in the Encounter Data Activity	
ISA06	Sender ID - Value set in the Encounter Data Activity	
ISA07	Receiver Qual - Value set in the Encounter Data Activity	
ISA08	Receiver ID - Value set in the Encounter Data Activity	
ISA09	Current Datestamp	
ISA10	Current Timestamp	
ISA11	Value set in the Encounter Data Activity	
ISA12	Value set in the Encounter Data Activity	
ISA13	ICN - Value set in the Encounter Data Activity, right padded with zeros	
ISA14	0	
ISA15	P	

ISA16	Value set in the SegPool
GS01	Value set according to specification
GS02	Sender Code - Value set in Encounter Data Activity
GS03	Receiver Code - Value set in Encounter Data Activity
GS04	Current Datestamp
GS05	Current Timestamp
GS06	GCN - Value set in the Encounter Data Activity
GS07	X
GS08	Value set according to specification
ST02	TCN - Value set in the Encounter Data Activity
ST03	Value set according to specification
BHT04	Current Datestamp
BHT05	Current Timestamp
SE01	Calculated Segment Count
SE02	TCN - Value set in the Encounter Data Activity
GE01	Calculated Transaction Count
GE02	GCN - Value set in the Encounter Data Activity
IEA01	Calculated Group Count
IEA02	ICN - Value set in the Encounter Data Activity
HL01	Indexed according to situation
HL02	Indexed according to situation
HL03	20,21,22, or 23
HL04	Chiapas Gate will fill in for a 1 for when HL03=20 and a 0 for when HL03=23. User must supply 0 or 1 according to the presence of children HL loops.
LX01	Incremented according to situation

When the Encode operation successfully completes, the SegPool register will be set to a new set of segments and elements. The specific rules for creating valid EDI files are covered in the HIPAA

Implementation Guides and are beyond the scope of this manual; the thirteen vw_Export_* views bundled with the distribution database can be used as a starting point of the minimal data elements needed to create a compliant transaction.

	CGateXltDecode	
	Parameters	None
Error Messages	<p>Data present on an element marked as Unused</p> <p>Element x (data) Mismatch with Data Type (AN 1/80)</p> <p>Missing qualifier for an identifier</p> <p>This element is not a valid value for this element</p> <p>This element is not a valid value for this element</p> <p>Data not present on an element marked as Required</p> <p>Data not present on an element marked as Required</p> <p>Premature End of File Detected</p> <p>Mandatory Segment Loop Name / S01 SegName Not Found</p> <p>Mandatory Loop Loop Name Not Found</p> <p>Segment LoopName / SegmentName Exceeded Maximum Iterations (max)</p> <p>Loop LoopName Exceeded Maximum Iterations (iterations)</p> <p>Insufficient segments found within source X12 file.</p> <p>Insufficient elements in the Group Envelope GS segment. This is not a HIPAA 4010/5010 file.</p> <p>The Element Separator, Segment Separator and Composite Element Separator characters defined in the ISA segment must all be different.</p> <p>ChiapasGate could not determine the specification from the information within the file.</p> <p>Premature End of File</p> <p>Unknown specification encountered.</p> <p>Exception during parsing</p> <p>Composite element exceeds that maximum allowed by the HIPAA Implementation Guide</p>	

Purpose:

This activity translates a loaded SegPool register to the HKey register. The decoding engine will flag for all errors it encounters but will not stop the decode process unless there was a critical syntax error that prevented the decoder engine from parsing the remainder of the file. Orphaned qualifier/identifier pairs will be flagged as an error and then ignored if they are not critical to decoding.

The decoder engine will detect and flag Level 1 and 2 SNIP integrity errors. Parse errors will be flagged as a critical error; care should be taken to poll the CRIT_ERR value from within the CGateQueryState activity to ensure that the workflow does not attempt to do further work on an HKey that was not loaded due to the critical error.

	CGateCNNAdd	
	Parameters	CNNDataSource (string) CNNDBType (string) CNNName (string)
Error Messages	CGateCNNAdd Exception:	
Purpose: This activity registers a named data source with the state object associated with the CGateContext activity. Other CGate I/O activities can refer to this connection by name alone. The CNNDataSource parameter contains the connection string, the DB Type contains the database type (“SQL Server”) and the connection is associated with the given name. Database connections should be opened after they are defined here, used and then closed. There is a default, open connection that does not need to be defined, opened or closed: TriggerDB. This points to the chiapasData database and is available in every workflow.		

	CGateCNNOpen	
	Parameters	CNNName (string)
Error Messages	CGateCNNOpen Exception: This does not contain the specified named connection There was an exception raised while opening the named connection	
Purpose: This opens the specified connection.		

	CGateCNNClose	
	Parameters	CNNName (string)
Error Messages	CGateCNNClose Exception: This does not contain the specified named connection There was an exception raised while closing the named connection	
Purpose: This closes the specified connection. All opened connections should be closed before the completion of the workflow.		

	CGateCNNSQLExec	
	Parameters	CNNName (string) CNNSQL (string)
Error Messages	CGateCNNSQLExec Exception: This does not contain the specified named connection There was an exception raised while executing the SQL on the named connection Retrying SQL:	
Purpose: This command will execute an arbitrary SQL command (as defined in CNNSQL) on the specified connection. This activity has been optimized for concurrent execution (with potentially dozens of workflows executing SQL commands and contending for resources simultaneously) in the following ways: <ul style="list-style-type: none">- Automatic Rollbacks – Any SQL command that leaves the connection in an uncommitted transaction will be automatically rolled back. As a result, all SQL Execution activities need to be transaction isolated – they cannot begin a transaction in one activity and then COMMIT it another activity. This prevents deadlocked SQL processes from leaving a lock on SQL Server resources.- If the SQL command raises an unhandled exception, the activity will retry the SQL until it does not raise an exception or for four additional times, whichever comes first. Each retry attempt will be flagged as an error in the MSGLOG along with the source code of the SQL statement.		

These features increase the reliability of executing SQL Server stored procedures in a high load environment. If there are unhandled exceptions within the SQL code being executed, however, be aware that this code could be consecutively re-executed for up to five iterations.

	CGateCNNSQLExecScript	
	Parameters	CNNName (string) CNNSQL (string)
Error Messages	CGateCNNSQLExecScript Exception: This does not contain the specified named connection There was an exception raised while executing the SQL on the named connection Retrying SQL Script:	
Purpose: This command loads a SQL text file from disk and executes it. It follows the same conventions as the CGateCNNSQLExec command.		

	CGateCNNSQLGetScalar	
	Parameters	CNNName (string) CNNSQL (string) Result (object)
Error Messages	CGateCNNSQLGetScalar Exception	
Purpose: This activity will execute a SQL query against a connection and then store the result into the variable specified in the Result parameter. The variable should be of type 'object'. Like other SQL Execution functions, it will attempt the scalar query up to five times. If all of these attempts result in a SQL Exception, the returned value will be set to null. If the query is successful but returns a database null value, the object variable will be set to: System.DBNull.Value.		

	CGateCNNFlatToTable	
	Parameters	CNNName (string) CNNReplace (string) CNNTTable (string)
Error Messages	CGateCNNFlatToTable Exception: This does not contain the specified named connection There was an exception raised while transferring the flat object to the table on the named connection	
Purpose: This activity exports the contents of the Flat register to the indicated table on the named connection. If the CNNReplace variable is set to lowercase ‘true’, the table will be dropped prior to data insertion. If the table does not exist, it will be created first. Then, all Flat rows will be transformed to SQL Insert statements executed on the destination table. If the CNNReplace flag is not set, this activity will generate an error if fields within the Flat register are not present in the destination table.		

	CGateCNNFlatFromTable	
	Parameters	CNNName (string) CNNTTable (string)
Error Messages	CGateCNNFlatFromTable Exception: This does not contain the specified named connection There was an exception raised while transferring the table/view to the internal Flat object	
Purpose: This activity imports the Flat register from the specified SQL table on the indicated connection. The CNNTTable may be a table name, a SELECT statement, or a stored procedure if the string begins with EXEC. If the stored procedure executed leaves an uncommitted transaction, it will be rolled back and flagged as an error.		

	CGateCNNFetchDT	
	Parameters	CNNName (string) CNNTTable (string) Result (System.Data.DataTable)
Error Messages	CGateCNNFetchDT Exception: This does not contain the specified named connection There was an exception raised while obtaining a DataTable from the named connection	
Purpose: This activity will load a table, SQL SELECT statement or stored procedure (preceded by EXEC in all caps) supplied in the CNNTTable parameter and set it to the System.Data.DataTable variable supplied in the Result field.		

	CGateCNNPutDT	
	Parameters	CNNName (string) DT (System.Data.DataTable) CNNTTable (string) Replace (string)
Error Messages	CGateCNNPutDT Exception: This does not contain the specified named connection There was an exception raised while storing the datatable to the specified named connection Limiting SQL table to first 1024 columns	
Purpose: This activity will translate a supplied DataTable into a series of INSERT statements. If the supplied table does not exist, it will be created. If the Replace parameter is set to 'true' (lowercase), the table will first be dropped and recreated. If the supplied DataTable contains more than 1024 columns, the output will be limited to only the first 1024 columns.		

	CGateDTToFlat	
	Parameters	DT (System.Data.DataTable)
Error Messages	CGateDTToFlat Exception:	
Purpose: This activity provides a way to programmatically setting the Flat register directly from a DataTable variable. The DataTable must contain string fields named according to the conventions of the CGIF Flat format.		

	CGateDTFromFlat	
	Parameters	DT (System.Data.DataTable)
Error Messages	CGateDTFromFlat Exception:	
Purpose: This activity copies the Flat register directly to a supplied DataTable variable.		

	CGateHKeyToXML	
	Parameters	HKeyXML (string)
Error Messages	CGateHKeyToXML Exception: Exception while transferring XmlDocument to HKey	
Purpose: This activity copies the HKey register to a supplied string variable using the CGIFXML2 format.		

	CGateHKeyFromXML	
	Parameters	HKeyXML (string)
Error Messages	CGateHKeyFromXML Exception: Exception while transferring XmlDocument to HKey	
Purpose: This activity loads the HKey register from a supplied string variable using the CGIFXML2 format.		

	CGateQueryState	
	Parameters	Query (string) Result (string)
Error Messages	CGateQueryState Exception:	

Purpose:

This activity allows you to directly poll various internal registers and variables belonging to the Chiapas EDI state object that is associated with the containing CGateContext activity. The following table describes the query string to be set on the Query field and the expected return value that will be stored in the variable supplied in the Result field:

Query String	Return Value
EVENT_ID	Automation EVENT_ID that spawned this workflow execution
SEGTERM	Segment Terminator character of the current SegPool object
ELETERM	Element Terminator character of the current SegPool object
SUBELETERM	Sub-Element Terminator of the current SegPool object
SPEC_SHRT_NM	Short name of the specification set on the State register. This is affected by load operations on the Flat or HKey registers. 4010_270 4010_271 4010_276 4010_277 4010_278_REQ 4010_278_RESP 4010_820 4010_834 4010_835 4010_837D 4010_837I 4010_837P 5010_270 5010_271 5010_276 5010_277 5010_277CA 5010_278_REQ 5010_278_RESP 5010_820 5010_820X 5010_834 5010_835 5010_837D 5010_837I 5010_837P

SPEC_LONG_NM	<p>Long name of the currently loaded specification. This is affected by load operations on the Flat or HKey registers.</p> <p>4010 Health Care Eligibility Benefit Inquiry</p> <p>4010 Health Care Eligibility Benefit Response</p> <p>4010 Health Care Claim Status Request</p> <p>4010 Health Care Claim Status Response</p> <p>4010 Health Care Services Review - Request</p> <p>4010 Health Care Services Review - Response</p> <p>4010 Payroll Deducted and Other Group Premium Payment for Insurance Products</p> <p>4010 Benefit Enrollment and Maintenance</p> <p>4010 Health Care Claim Payment/Advice</p> <p>4010 Health Care Claim: Dental</p> <p>4010 Health Care Claim: Institutional</p> <p>4010 Health Care Claim: Professional</p> <p>5010 Health Care Eligibility Benefit Inquiry</p> <p>5010 Health Care Eligibility Benefit Response</p> <p>5010 Health Care Claim Acknowledgment</p> <p>5010 Health Care Claim Status Request</p> <p>5010 Health Care Claim Status Response</p> <p>5010 Health Care Services Review - Request for Review</p> <p>5010 Health Care Services Review - Response</p> <p>5010 Payroll Deducted and Other Group Premium Payment for Insurance Products</p> <p>5010 820X Health Insurance Exchange Related Payments</p> <p>5010 Benefit Enrollment and Maintenance</p> <p>5010 Health Care Claim Payment/Advice</p> <p>5010 Health Care Claim: Dental</p> <p>5010 Health Care Claim: Institutional</p> <p>5010 Health Care Claim: Professional</p>
CRIT_ERR	This returns 'true' if the critical error flag has been set on the ErrLog register
ERR_CT	Count of ErrLog messages
ERR_LOG	All ErrLog messages compiled into a single string value
FLAT_COL_CT	Number of columns present within the Flat register
FLAT_ROW_CT	Number of rows present within the Flat register
SEG_CT	Number of segments present within the SegPool register
FLAT_SPEC_NM	

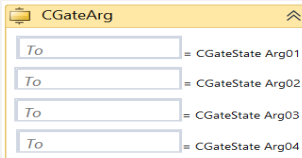
HKEY_SPEC_NM

SEG_SPEC_NM

These query strings will interrogate the currently loaded specification string associated with each of the three registers after a load operation.

Return Value	Meaning
4010_270.BIN	Health Care Eligibility Benefit Inquiry (270)
4010_270_A1.BIN	Addenda 1
4010_271.BIN	Health Care Eligibility Benefit Inquiry Response (271)
4010_271_A1.BIN	Addenda 1
4010_276.BIN	Health Care Claim Status Request (276)
4010_276_A1.BIN	Addenda 1
4010_277.BIN	Health Care Claim Status Response (277)
4010_277_A1.BIN	Addenda 1
4010_278_REQ.BIN	Health Care Services Review - Request for Review (278)
4010_278_REQ_A1.BIN	Addenda 1
4010_278_RESP.BIN	Health Care Services Review - Response (278)
4010_278_RESP_A1.BIN	Addenda 1
4010_820.BIN	Payroll Deducted and Other Group Premium Payment for Insurance Product (820)
4010_820_A1.BIN	Addenda 1
4010_834.BIN	Benefit Enrollment and Maintenance (834)
4010_834_A1.BIN	Addenda 1
4010_835.BIN	Health Care Claim Payment / Advice (835)
4010_835_A1.BIN	Addenda 1
4010_837D.BIN	Health Care Claim: Dental (837)
4010_837D_A1.BIN	Addenda 1
4010_837I.BIN	Health Care Claim: Institutional (837)
4010_837I_A1.BIN	Addenda 1
4010_837P.BIN	Health Care Claim: Professional (837)
4010_837P_A1.BIN	Addenda 1
5010_270.BIN	Health Care Eligibility Benefit Inquiry (270)
5010_270_A1.BIN	Addenda 1
5010_271.BIN	Health Care Eligibility Benefit Inquiry Response (271)
5010_271_A1.BIN	Addenda 1
5010_276.BIN	Health Care Claim Status Request (276)
5010_277.BIN	Health Care Claim Status Response (277)
5010_277CA.BIN	Health Care Claim Acknowledgment (277CA)
5010_278_REQ.BIN	Health Care Services Review - Request for Review (278)
5010_278_RESP.BIN	Health Care Services Review - Response (278)
5010_820.BIN	Payroll Deducted and Other Group Premium Payment for Insurance Products (820)
5010_820X.BIN	Health Insurance Exchange Related Payments
5010_820_A1.BIN	Addenda 1
5010_824.BIN	Application Reporting for Insurance (824)
5010_834.BIN	Benefit Enrollment and Maintenance (834)
5010_834_A1.BIN	Addenda 1
5010_835.BIN	Health Care Claim Payment / Advice (835)
5010_835_A1.BIN	Health Care Claim Payment / Advice (835) A1
5010_837D.BIN	Health Care Claim: Dental (837)
5010_837D_A1.BIN	Addenda 1
5010_837D_A2.BIN	Addenda 2
5010_837I.BIN	Health Care Claim: Institutional (837)
5010_837I_A1.BIN	Addenda 1
5010_837I_A2.BIN	Addenda 2
5010_837P.BIN	Health Care Claim: Professional (837)
5010_837P_A1.BIN	Addenda 1

	CGateClearRegister											
	Parameters	Register (string)										
Error Messages	CGateClearRegister Exception:											
Purpose: This activity enables you to clear loaded registers. The Register argument must contain one of the following values:												
<table><tr><th>Register Value</th><th>Purpose</th></tr><tr><td>SEGP00L</td><td>Clears the SegPool register</td></tr><tr><td>HKEY</td><td>Clears the HKey register</td></tr><tr><td>FLAT</td><td>Clears the Flat register</td></tr><tr><td>ALL</td><td>Clears all three registers</td></tr></table>			Register Value	Purpose	SEGP00L	Clears the SegPool register	HKEY	Clears the HKey register	FLAT	Clears the Flat register	ALL	Clears all three registers
Register Value	Purpose											
SEGP00L	Clears the SegPool register											
HKEY	Clears the HKey register											
FLAT	Clears the Flat register											
ALL	Clears all three registers											
This is helpful when you need an added degree of memory management control and have no need of certain registers after a certain point in your workflow.												

	CGateArg	
	Parameters	To1 (string) To2 (string) To3 (string) To4 (string)
Error Messages		
Purpose: This activity enables an easy way to transfer information from the automation system to a running workflow. When a new event is created, there are four columns available to hold argument data, EVENT_DATA01, EVENT_DATA02, EVENT_DATA03, and EVENT_DATA04 (varchar (200) data type). When a user places a string variable within these fields, whatever value was present within the column will be propagated to the string variable. See the EVENTS table schema for more information about the values generated by the automation system when it generates new events.		

	CGateMerge	
	Parameters	DestCnn (string) DestTable (string)
Error Messages	CGateMerge Exception: The Merge command is only functional for SQLite and SQL Server connections Unmerged Mapping:	
Purpose: This activity is a more advanced method for storing a Flat register to a database table. Unlike the CGateFlatToTable activity or the CGateTransfer activity, this will do a schema analysis on the destination and insert only those fields that both the Flat register and the destination table have in common. Fields that are present within the Flat Register but not in the destination table will result in 'Unmerged Mapping:' errors added to the ErrLog. The DestCnn connection string may be either 'TriggerDB' or the name of an opened database connection. The DestTable is the name of an existing database table.		

	CGateForceMerge	
	Parameters	DestCnn (string) DestTable (string)
Error Messages	CGateForceMerge Exception: Unmerged Mapping (Max Columns Reached):	
Purpose: This activity is functionally similar to the CGateMerge activity with one crucial difference – instead of generating messages when there are Flat fields missing from the destination table, it will actually carry out ETL data operations to add the missing fields to the destination table prior to the INSERT statements. This guarantees that the destination tables will always be able to completely store incoming transactions, but also means that the destination table schema is not fixed and could grow over time. Only the first 1020 columns will be merged, as SQL Server is limited to 1024 columns. DestCnn connection string may be either 'TriggerDB' or the name of an open database connection. DestTable is the name of a table that must already exist.		

	CGateXFEROpenSession	
	Parameters	SessionName (string)
Error Messages	Failed to Open Named Session	
Purpose: This activity opens a previously setup WinSCP session. Only one session can be open at a time. The session name should be copied directly from the session as shown within the WinSCP manager (C:\chiapas\bin\winscp.exe). When saving the session, make sure to select the option to ‘Save Password’ so that the session has all the information necessary to log into the SecureFTP server. The protocol will be defaulted to SecureFTP. In order to utilize WinSCP sessions configured for FTP, add a ‘FTP:’ prefix here to the SessionName argument (note: this prefix should <i>not</i> be present within the actual WinSCP session name, it is only used to prefix the session name when using this activity). NOTE: FTP is not a natively encrypted protocol. Sending unencrypted PHI through the Internet is a critical HIPAA violation, so exercise extreme caution when using this option.		

	CGateXFERCloseSession	
	Parameters	None
Error Messages	Session is already closed	
Purpose: This activity closes an opened WinSCP session. This should be placed in the Finally clause of a Try/Catch/Finally activity to ensure that all opened sessions are properly closed, even if an exception occurs during an operation.		

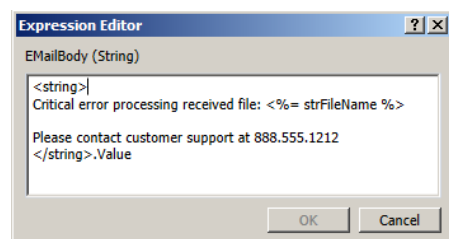
	CGateXFERPutFile	
	Parameters	FileName (string) RemoteFileName (string)
Error Messages	Session is not open - XFER_PutFile failed Error with PutFile: Exception during PutFile operation	
Purpose: This activity will upload a local file (in FileName) to a remote directory with the filename specified in 'RemoteFileName'. All transfers will take place in BINARY mode.		

	CGateXFERGetFile	
	Parameters	FileName (string) RemoteFileName (string)
Error Messages	Session is not open - XFER_PutFile failed Error with GetFile: Exception during GetFile operation	
Purpose: This activity downloads a remote file (RemoteFileName) to the local filesystem with the given filename. All transfers will take place in BINARY mode.		

	CGateXFERDeleteFile	
	Parameters	RemoteFileName (string)
Error Messages	Session is not open - XFER_DeleteFile failed Deletion Failure: Delete Failure - file doesnt exist Delete Failure	
Purpose: This activity deletes the file indicated in RemoteFileName from the remote SecureFTP server.		

	CGateXFERGetDirectoryFiles	
	Parameters	RemoteDirectory (string) FileList (string[])
Error Messages	CGateXFERGetDirectoryFiles Exception:	
Purpose: This activity retrieves a directory of files in the specified directory and records the filenames in the specified string[] variable (FileList).		

	<p>CGateSendMail</p> <table border="1"> <thead> <tr> <th>Parameters</th><th></th></tr> </thead> <tbody> <tr> <td>EMailBody (string)</td><td></td></tr> <tr> <td>EMailReceiver (string)</td><td></td></tr> <tr> <td>EMailSender (string)</td><td></td></tr> <tr> <td>EMailSubject (string)</td><td></td></tr> <tr> <td>SMTP_Host (string)</td><td></td></tr> <tr> <td>SMTP_Port (integer)</td><td></td></tr> <tr> <td>SMTP_SSL (string)</td><td></td></tr> <tr> <td>SMTP_Username (string)</td><td></td></tr> </tbody> </table>	Parameters		EMailBody (string)		EMailReceiver (string)		EMailSender (string)		EMailSubject (string)		SMTP_Host (string)		SMTP_Port (integer)		SMTP_SSL (string)		SMTP_Username (string)	
Parameters																			
EMailBody (string)																			
EMailReceiver (string)																			
EMailSender (string)																			
EMailSubject (string)																			
SMTP_Host (string)																			
SMTP_Port (integer)																			
SMTP_SSL (string)																			
SMTP_Username (string)																			
Error Messages	<p>Error with SMTP Activity (Subject):</p> <p>Error with SMTP Activity (Message Body):</p> <p>Error with SMTP Activity (Sender):</p> <p>Error with SMTP Activity (Receiver):</p> <p>Error with SMTP Activity (creating network credentials):</p> <p>Error with SMTP Activity (getting server):</p> <p>Error with SMTP Activity (SSL argument must be set to true or false):</p> <p>Error with SMTP Activity (error on sending message):</p>																		
<p>Purpose:</p> <p>This activity will open a connection to an SMTP (SendMail) server and send out an Email with the specified login credentials.</p> <p>EMailBody:</p> <p>The main text of the Email. A string variable or expression can be supplied, or a tagged XML expression such as the following can be used to easily embed variables:</p> <pre><string> Critical error processing received file: <%= strFileName %> Please contact customer support at 888.555.1212 </string>.Value</pre> <p>If the expression resolves to HTML, an HTML-formatted Email will be transmitted.</p> <p>EmailReceiver:</p> <p>This string should contain at least one valid Email address. Multiple Email addresses can be specified by separating them with a semicolon.</p> <p>EMailSender:</p> <p>The Email address of the sending party.</p>																			



EMailSubject:

Subject of the Email.

SMTP_Host:

Server name or IP address of the SendMail server.

Example: smtp.gmail.com

SMTP_Password:

Password used for the SendMail login account supplied in SMTP_Username.

SMTP_Port:

Server port to connect to. Common values are 25 or 587.

SMTP_SSL:

This should be set to "true" for SSL encrypted SMTP logins, "false" otherwise.

SMTP_Username:

Login credential used when logging into the SendMail server.

	CGateIOCopyFile	
	Parameters	CopyFrom (string) CopyTo (string)
Error Messages	Exception during CGateIOFileCopy:	
Purpose: This activity copies the file specified in CopyFrom and copies it to the path and filename indicated in the CopyTo parameter. This activity can be combined with the CGateIODeleteFile to move a file.		

	CGateIODeleteFile	
	Parameters	DeleteFile (string)
Error Messages	Exception during CGateIODeleteFile:	
Purpose: This activity deletes the specified file.		

	CGateIODirectory	
	Parameters	DirectoryPath (string) ResultFiles (string[])
Error Messages	Exception during CGateIOEnumDirectory:	
Purpose: This activity retrieves the list of files in the given directory to the string array variable.		

	CGateRunShell	
	Parameters	ShellCmd (string) ShellArgs (string)
Error Messages	CGateRunShell Exception: Exit Code:	
Purpose: This activity will execute an arbitrary shell command and wait for its completion. If the command returns a non-zero exit code, a message is added to the ErrLog indicating the Exit Code. ShellCmd – A string representing the full path and filename of the shell command to execute ShellArgs – A string storing any command line arguments for the shell command		

	CGateRunShell2	
	Parameters	ShellCmd (string) ShellArgs (string) ShellOutput (string)
Error Messages	CGateRunShell2 Exception: Exit Code:	
Purpose: This activity will execute an arbitrary shell command and wait for its completion. If the command returns a non-zero exit code, a message is added to the ErrLog indicating the Exit Code. If there is any console output, it will be stored in the variable provided in the ShellOutput box. ShellCmd – A string representing the full path and filename of the shell command to execute ShellArgs – A string storing any command line arguments for the shell command ShellOutput – A string for storing any console output generated by the shell command.		

SOLUTION WORKBOOK

Up to this point, the manual has covered the theory and operation behind the different components of the platform. In this section, we will review actually putting them together to fulfill actual business requirements and in so doing, exercise different aspects of the technology. This workbook will describes a single complex business scenario, and starts with detailed business requirements. This is followed up with step-by-step instructions following the path of 'discovery' to fulfill these business requirements, including testing and correcting problems that occur along the way.

The example solution relies heavily on SQL Server and therefore you should have SQL Server Management Studio available.

SOLUTION: Legacy Data to 837P Claim EDI Files

Business Requirements

The *George Pickle Clinic* is a medium sized healthcare services provider in Anytown, USA, founded in the 1970's. GP Clinic submits professional claims to the Fly-by-Night HMO for all its product lines. However, FBN HMO has mandated that legacy text files will not be accepted after an internal deadline in order to become HIPAA compliant. Furthermore, FBN HMO charges a \$5 surcharge for processing all paper claims, which is not a financially viable option with the normally lower cost services GP Clinic provides.

GP Clinic runs off a UNIX based mainframe, and contains all of GP Clinics patient and billing information. The closed, proprietary architecture is not especially amenable to connecting to a database, but the legacy process of creating Comma Separated Values files containing necessary billing information is well established and has been the cornerstone of billing to FBN HMO.

GP Clinic needs to create 837P compliant EDI files according using the legacy CSV files it has been generating and then Email the EDI coordinator when the file is available.

A successful implementation would meet the following criteria:

1. The Legacy CSV process is the sole data source, and should be mapped in such a way as to create a compliant 837P file.
2. Once the Legacy CSV file is dropped into the input directory, it should automatically push the converted EDI file to the outbound directory ready to be uploaded to FBN HMO.

CSV Specifications

The Legacy CSV Process consists of a CSV file containing the following mappings. Each row represents one claim with up to four service lines. If a claim contains more than four service lines, the claim header information is repeated. All patients are considered subscribers in FBN HMO so there is no need for transmitting patient data separately from subscriber data. No header row is transmitted in this format so the column names are for reference only. All fields are quote delimited.

Column Name	Description and Possible Values
FILE_DATE	Timestamp the file was generated in the format YYYY-MM-DD Example: 2014-07-04
LAST_NAME	Patient Last Name
FIRST_NAME	Patient First Name
SSN	Patient Social Security Number in the format NNN-NN-NNNNN Example: 555-55-5555
ADDRESS	Patient Home Address
CITY	Patient City
STATE	Patient State
ZIP_CODE	Patient ZIP Code
BIRTH_DATE	Patient Date of Birth in the format YYYY-MM-DD Example: 1939-04-05
SEX	Patient Gender – M for Male, F for Female
CLAIM_ID	Claim Identifier – a 10 digit number
CLINIC_ID	Clinic Location Identifier – a 1 digit number indicating the clinic where services were rendered: 1 – PB Clinic Felix Location, 101 Cat Street, Anytown, CA 00001 2 – PB Clinic Jelly Facility, 303 Peanut Butter Ave, Anytown, CA 00001 3 – PB Clinic Unlit Lab, 1000 Mad Scientist Way, Anytown, CA 00001
DX	Diagnosis Code with ‘Dot’ Code Example: V72.83
SERVICE_CODE_1	HCPCS Service Code for Service Rendered Example: 99213
SERVICE_DATE_1	Date Service was Rendered in the format YYYY-MM-DD Example: 2014-07-06
SERVICE_CHARGE_1	Service Charge Amount in the format NNNN.NN Example: 1234.56
SERVICE_CODE_2	Second Service Rendered
SERVICE_DATE_2	
SERVICE_CHARGE_2	

SERVICE_CODE_3	Third Service Rendered
SERVICE_DATE_3	
SERVICE_CHARGE_3	
SERVICE_CODE_4	Fourth Service Rendered
SERVICE_DATE_4	
SERVICE_CHARGE_4	

CSV Sample

```
"2014-07-06","STEW","MARNEY","111-22-9999","976 GINGERBREAD LN. APT
6","ANYTOWN","CA","00001","1962-03-01","F","8980153154","1","V72.83","99213","2014-
07-01","42.00","","","","","","","","",""
"2014-07-06","KNIFE","MACK","111-22-9000","123 WHITTLEBIRD
ST","ANYTOWN","CA","00001","1973-07-16","M","8980153155","2","302.89","55250","2014-
07-01","205.00","","","","","","","","",""
"2014-07-06","SUNDAE","DEVIN","111-22-8000","842 TURBAN
DR","ANYTOWN","CA","00001","1970-01-14","M","8980153156","3","279.00","80048","2014-
07-02","15.00","80076","2014-07-02","15.00","80202","2014-07-
02","17.00","85025","2014-07-02","10.00"
"2014-07-06","SUNDAE","DEVIN","111-22-8000","842 TURBAN
DR","ANYTOWN","CA","00001","1970-01-14","M","8980153156","3","279.00","85652","2014-
07-02","12.00","","","","","","","","",""
```

Technical Analysis

Requirement 1	The Legacy CSV process is the sole data source, and should be mapped in such a way as to create a compliant 837P file.
Assessment	<p>This requirement is basically stating that the CSV file is the sole input data source from the legacy environment; since this has been used successfully as the production billing data feed in the past, it should therefore have all the data necessary for a valid EDI file.</p> <p>A close look at the 837P EDI requirements shows the following steps will be needed to fulfill the requirement.</p>
Deliverable 1-A Outer Loops / ICN	The ISA, GS and ST envelope loops will all need to be given default values. There needs to be a mechanism implemented to generate a new, incrementing ISA13 Interchange Control Number for each transaction.
Deliverable 1-B 1000A 1000B	Fixed values for the 1000A Submitter Loop and 1000B Receiver Loop need to be established and mapped.
Deliverable 1-C 2010AA	Values for the Billing Provider 2010AA need to be established and mapped.
Deliverable 1-D 2000B	Values for 2000B SBR segment need to be established and mapped

Deliverable 1-E 2010BA	The data fields for LAST_NAME, FIRST_NAME, SSN, ADDRESS, STATE, CITY, ZIP_CODE, BIRTH_DATE and SEX need to be mapped to the correct elements in the 2010BA Subscriber loop. All values will need to be in the correct format for EDI files.
Deliverable 1-F 2300	The Implementation Guide states a number of Claim Header fields are required. These can be defaulted according to certain business practices; the fields CLAIM_ID, CLINIC_ID and DX will need to be mapped to the 2300 loop, while allowing for the possibility that the same claim could repeat on multiple lines.
Deliverable 1-G 2400	SERVICE_CODE_1, SERVICE_DATE_1, SERVICE_CHARGE_1 will need to be mapped to Loop 2400 / Service Line, with the remaining three service field groups being mapped to additional service lines. Because Service Lines can span multiple source file rows,
Requirement 2	Automation should be in place to translate the CSV file to the new format.
Assessment	This requirement can be broken down into two discrete sets of deliverables. First, we will need to set up a new trigger to begin the workflow when a new CSV file appears in a certain directory. This will be set up as a LOCAL UPLOAD trigger within the Chiapas EDI environment. Second, the file should be translated to EDI as described in Requirement 1 and placed in another directory.
Deliverable 2-A	<p>We will assume that the PB Clinic IT department is able to get the legacy CSV to the Chiapas EDI server directory:</p> <p>C:\chiapas\pipeline\013_CSVLegacy\01_in_csv</p> <p>A new trigger will be set up to move the file to:</p> <p>C:\chiapas\pipeline\013_CSVLegacy\02_in_csv</p> <p>Then, a SQL Server BULK INSERT process will propagate this raw file to an internal database table on the chiapasData database.</p>
Deliverable 2-B	The raw file should be inserted to a table called BIN_FBN and follow the schema of the CSV file.
Deliverable 2-C	<p>The data will then be translated and pushed into the BIN_837P using a new BIN_ID. Then, a new trigger is established to create a new outbound EDI file in the following directory.</p> <p>C:\chiapas\pipeline\013_CSVLegacy\03_out_edi</p>

Deliverable 1	<p>As Chiapas EDI Enterprise already contains an 837P EDI export path, it is far easier to alter the existing ETL scripts than it is to create all of them from scratch. Examining the vw_Export_837P view, it appears to draw many defaulted values from the SAMPL_TRAD_PARTNER and SAMPL_PROVIDER tables. Our first step would then be to replicate these tables to production tables and insert the new defaults according to the business requirements.</p> <p>First, from within SSMS, we create a new version of the tables:</p> <pre>SELECT TOP 0 * INTO PROD_TRAD_PARTNER FROM SAMPL_TRAD_PARTNER SELECT TOP 0 * INTO PROD_PROVIDER FROM SAMPL_PROVIDER</pre>
In order to satisfy the deliverables, we will need to create entries for the following:	
<p>Table: PROD_TRAD_PARTNER</p> <p>1 row describing the elements for a transaction from PB Clinic to FBN HMO.</p> <pre>INSERT INTO PROD_TRAD_PARTNER SELECT 'FBN HMO UPLOAD' TP_NAME, 1 SENDR_PROV_ID, 2 RECVR_PROV_ID, SPACE(10) ISA02, SPACE(10) ISA04, 'GP_CLINIC_ID' + SPACE(3) ISA06_SENDR_ID, 'FBN_HMO_ID' + SPACE(5) ISA08_RECVR_ID, '^' ISA11_REPTN_SEP, 1 ISA14_ACK_REQ, 'P' ISA15_USG_IND, ':' ISA16_SUBELE_SEP, 100000000 ICN, 1 GCN, 100000000 TCN, 'GP_CLINIC_ID' GS02_APP_SENDR_CD, 'FBN_HMO_ID' GS03_APP_RECVR_CD</pre>	
<p>Table: PROD_PROVIDER</p> <p>5 rows describing the rows for all of the entities involved: One each for the sender and receiver (GP Clinic and FBN HMO), and three additional providers to represent the three sites that are generating claims. The autonumber will automatically assign a PROV_ID to each of the rows.</p> <pre>INSERT INTO PROD_PROVIDER SELECT '999488801' TAX_ID, '123400001' NPI_ID, 'GEORGE PICKLE CLINIC' COMPANY_NM, NULL FIRST_NM,</pre>	

NULL	LAST_NM,
'4000 MAIN ST.'	ADD1,
'SUITE 201'	ADD2,
'ANYTOWN'	CITY,
'CA'	STATE,
'00001'	ZIP_CD,
'GEORGE PICKLE JR'	CONTACT_NM,
'4155551212'	WORK_PH,
NULL	EMAIL UNION SELECT
'999488802'	TAX_ID,
'123400002'	NPI_ID,
'FLY BY NIGHT HMO'	COMPANY_NM,
NULL	FIRST_NM,
NULL	LAST_NM,
'100 MAIN ST.'	ADD1,
NULL	ADD2,
'ANYTOWN'	CITY,
'CA'	STATE,
'00001'	ZIP_CD,
'MICHAEL NIGHT'	CONTACT_NM,
'4155551213'	WORK_PH,
NULL	EMAIL UNION SELECT
'999488801'	TAX_ID,
'123400001'	NPI_ID,
'GP FELIX LOCATION'	COMPANY_NM,
NULL	FIRST_NM,
NULL	LAST_NM,
'101 CAT ST.'	ADD1,
NULL	ADD2,
'ANYTOWN'	CITY,
'CA'	STATE,
'00001'	ZIP_CD,
NULL	CONTACT_NM,
NULL	WORK_PH,
NULL	EMAIL UNION SELECT
'999488801'	TAX_ID,
'123400001'	NPI_ID,
'GP JELLY LOCATION'	COMPANY_NM,
NULL	FIRST_NM,
NULL	LAST_NM,
'303 PEANUT BUTTER AVE'	ADD1,
NULL	ADD2,
'ANYTOWN'	CITY,
'CA'	STATE,
'00001'	ZIP_CD,
NULL	CONTACT_NM,
NULL	WORK_PH,
NULL	EMAIL UNION SELECT
'999488801'	TAX_ID,
'123400001'	NPI_ID,
'GP UNLIT LAB'	COMPANY_NM,
NULL	FIRST_NM,
NULL	LAST_NM,
'1000 MAD SCIENTIST WY'	ADD1,
NULL	ADD2,
'ANYTOWN'	CITY,
'CA'	STATE,
'00001'	ZIP_CD,

NULL	CONTACT_NM,
NULL	WORK_PH,
NULL	EMAIL

Table:

BIN_FBN

Before we can get further, we'll need to get the storage area for the legacy CSV file set up. Using that as a data source, we will have the data available to feed into a modified vw_Export_837P view.

Reviewing the existing BIN repositories, we observe they each have a BIN_ID foreign key to the BIN_LOG table. We will replicate that system for this data source by creating the BIN_FBN table with these fields as well as the fields indicated in the CSV specification:

```
CREATE TABLE BIN_FBN
( BIN_ID          INT,
  FILE_DATE       VARCHAR(200),
  LAST_NAME       VARCHAR(200),
  FIRST_NAME      VARCHAR(200),
  SSN             VARCHAR(200),
  ADDRESS         VARCHAR(200),
  CITY            VARCHAR(200),
  STATE           VARCHAR(200),
  ZIP_CODE        VARCHAR(200),
  BIRTH_DATE      VARCHAR(200),
  SEX             VARCHAR(200),
  CLAIM_ID        VARCHAR(200),
  CLINIC_ID       VARCHAR(200),
  DX              VARCHAR(200),
  SERVICE_CODE_1  VARCHAR(200),
  SERVICE_DATE_1  VARCHAR(200),
  SERVICE_CHARGE_1 VARCHAR(200),
  SERVICE_CODE_2  VARCHAR(200),
  SERVICE_DATE_2  VARCHAR(200),
  SERVICE_CHARGE_2 VARCHAR(200),
  SERVICE_CODE_3  VARCHAR(200),
  SERVICE_DATE_3  VARCHAR(200),
  SERVICE_CHARGE_3 VARCHAR(200),
  SERVICE_CODE_4  VARCHAR(200),
  SERVICE_DATE_4  VARCHAR(200),
  SERVICE_CHARGE_4 VARCHAR(200)
)
```

Next, we need to populate the table to aid us in making the transformations necessary to create the EDI view. Taking the CSV Sample text, save it as a text file called 'DATA.CSV', then use SQL Server Import to pull the sample data into the BIN_FBN table. Also, ensure all columns are mapped:

SQL Server Import and Export Wizard

Choose a Data Source
Select the source from which to copy data.

Data source: Flat File Source

Select a file and specify the file properties and the file format.

File name: C:\Users\Administrator\Desktop\DATA.csv Browse...

Locale: English (United States) ☐ Unicode

Code page: 1252 (ANSI - Latin I)

Format: Delimited

Text qualifier: "

Header row delimiter: (CR)(LF)

Header rows to skip: 0

☐ Column names in the first data row

Column Mappings

Source: C:\Users\Administrator\Desktop\DATA.csv

Destination: [dbo].[BIN_FBN]

☐ Create destination table Edit SQL...

☐ Delete rows in destination table ☐ Drop and re-create destination table

☒ Append rows to the destination table ☒ Enable identity insert

Mappings:

Source	Destination	Type	Nullable	Size	Precision	Scale
Column 0	FILE_DATE	varchar	<input checked="" type="checkbox"/>	200		
Column 1	LAST_NAME	varchar	<input checked="" type="checkbox"/>	200		
Column 2	FIRST_NAME	varchar	<input checked="" type="checkbox"/>	200		
Column 3	SSN	varchar	<input checked="" type="checkbox"/>	200		
Column 4	ADDRESS	varchar	<input checked="" type="checkbox"/>	200		
Column 5	CITY	varchar	<input checked="" type="checkbox"/>	200		
Column 6	STATE	varchar	<input checked="" type="checkbox"/>	200		
Column 7	ZIP_CODE	varchar	<input checked="" type="checkbox"/>	200		
Column 8	BIRTH_DATE	varchar	<input checked="" type="checkbox"/>	200		

Source column: Column 24 string [DT_STR] (50)

OK Cancel

View:

vw_Export_FBN

Deliverables 1 A / 1 B

At this stage, we now have data in the BIN_FBN table, the PROD_PROVIDER table and the PROD_TRAD_PARTNER table. Now, we will adapt the existing vw_Export_837P to tie it to these new data sources:

```
CREATE VIEW [dbo].[vw_Export_FBN]
AS
SELECT ISA02                [X1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO],
       ISA04                [OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO],
       ISA06_SENDR_ID       [OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID],
       ISA08_RECVR_ID       [OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID],
       ISA11_REPTN_SEP      [OEISA_S01_ISA11_REPTN_SEPRTR],
       '00501'              [OEISA_S01_ISA12_VERSN_NR],
       ICN                  [OEISA_S01_ISA13_ICN],
       ISA14_ACK_REQ        [OEISA_S01_ISA14_ACK_REQ],
       ISA15_USG_IND        [OEISA_S01_ISA15_USG_IND],
       ISA16_SUBELE_SEP     [OEISA_S01_ISA16_SUBELE_SEP],
       ICN                  [OEISA_S03_IEA02_ICN],
       GS02_APP_SENDR_CD    [GSHDR_S01_GS02_APP_SENDR_CD],
       GS03_APP_RECVR_CD    [GSHDR_S01_GS03_APP_RECVR_CD],
       GCN                  [GSHDR_S01_GS06_GCN],
```

```

'005010X222A1' [GSHDR_S01_GS08_VERS_ID_CD],
GCN [GSHDR_S03_GE02_GCN],
TCN [STHDR_S01_ST02_CONTRL_NR],
'005010X222A1' [STHDR_S01_ST03_IMPLMNTN_GUID_VERSN_NM],
TCN [STHDR_S06_SE02_TCN],
'0019' [STHDR_S02_BHT01_STRUCTR_CD],
'00' [STHDR_S02_BHT02_TS_PURPS_CD],
RIGHT(CONVERT(VARCHAR,RAND( (DATEPART(mm, GETDATE()) * 100000)+
(DATEPART(ss, GETDATE()) * 1000 )+DATEPART(ms, GETDATE()) ),6)
[STHDR_S02_BHT03_ORGNTR_APLCTN_TRANSCTN_ID],
'CH' [STHDR_S02_BHT06_CLM_ENCNR_ID],
SENDER_PV.COMPANY_NM [L1000A_S01_NM103_SUBMTR_LAST_ORG_NM_OVL_NONPRSN_ENTY],
SENDER_PV.TAX_ID [L1000A_S01_NM109_SUBMTR_ID_OVL_ETN_NR],
SENDER_PV.CONTACT_NM [L1000A_S02_PER02_CONTCCT_NM],
SENDER_PV.WORK_PH [L1000A_S02_PER04_COMM_NR_OVL_TELPHN],
RECEIVER_PV.COMPANY_NM [L1000B_S01_NM103_RECEVR_NM_OVL_NONPRSN_ENTY],
RECEIVER_PV.TAX_ID [L1000B_S01_NM109_PRIMARY_ID_OVL_ETN_NR]
FROM PROD_TRAD_PARTNER TP
INNER JOIN PROD_PROVIDER SENDER_PV ON TP.SENDER_PROV_ID = SENDER_PV.PROV_ID
INNER JOIN PROD_PROVIDER RECEIVER_PV ON TP.RECEVR_PROV_ID = RECEIVER_PV.PROV_ID

```

At this point, we need to stop and validate our work so far. With the Chiapas Studio interface, this is very straightforward and can be done in seconds. We simply open up the Studio, put vw_Export_FBN into the table text box under Flat->DB Table, press Execute, then press the Flat > HKey button under the middle HKey Tab, then press Encode, and we see this result:

```

[000001] ISA*00*          *00*          *ZZ*GP_CLINIC_ID  *ZZ*FBN_HMO_ID
*140708*0053*^*00501*100000000*1*P*:
[000002] GS*HC*GP_CLINIC_ID*FBN_HMO_ID*20140708*0053*1*X*005010X222A1
[000003] ST*837*100000000*005010X222A1
[000004] BHT*0019*00*150927*20140708*0053*CH
[000005] NM1*41*2*GEORGE PICKLE CLINIC*****46*999488801
[000006] PER*IC*GEORGE PICKLE JR*TE*4155551212
[000007] NM1*40*2*GP FELIX LOCATION*****46*999488801
[000008] SE*6*100000000
[000009] GE*1*1
[000010] IEA*1*100000000

```

The most important part of quickly developing EDI mappings is the ability to quickly *iterate*. In this case, we observe that the 1000B is not correct – instead of listing FBN HMO, it is listing information related to another clinical location. This is simply corrected by updating PROD_TRAD_PARTNER to the correct ID of 5:

```
UPDATE PROD_TRAD_PARTNER SET RECEIVER_PROV_ID = 5 WHERE TP_ID = 1
```

Then, we select the 'Reset All' to clear the state of the Chiapas Studio and re-run the above steps, and we observe that the Receiving Provider is now set correctly to FBN HMO.

View:

vw_Export_FBN

Deliverable 1-C / 1-D

Next, we must extend the view to encode proper values for the 2010 AA (Billing Provider) and SBR segment. We alter the view and insert the following lines prior to the FROM clause. The L2000A HL04 mapping allows the encoder to encode the 2000A loop – by the rules of the logic engine, every loop

must have at least one mapped element. Following this is the Billing Provider information, which is the same as the Submitting Provider. Note that the ZIP Code is padded with four 0's.

For the SBR subscriber segment, this describes the type of insurance for the subscriber. Since all patients in this file are subscribers in Fly-By-Night HMO's commercial product, then 'P' (Primary Subscriber), '18' (Relationship: Self) and 'CI' (Commercial Insurance) are the correct values to populate in this transaction.

```
'1' [L2000A_S01_HL04_HIERCHCL_CHILD_CD],
SENDER_PV.COMPANY_NM [L2010AA_S01_NM103_BILNG_PROV_ORGL_NM_OVL_NONPRSN_ENTY],
SENDER_PV.NPI_ID [L2010AA_S01_NM109_PROV_ID_OVL_CENTRS_MEDCR_MEDCD_SVCS],
SENDER_PV.ADD1 [L2010AA_S02_N301_BILNG_PROV_ADRS_LIN],
SENDER_PV.ADD2 [L2010AA_S02_N302_BILNG_PROV_ADRS_LIN],
SENDER_PV.CITY [L2010AA_S03_N401_BILNG_PROV_CITY_NM],
SENDER_PV.STATE [L2010AA_S03_N402_STAT_PROVNC_CD],
CONVERT(VARCHAR,SENDER_PV.ZIP_CD) + '0000'
SENDER_PV.TAX_ID [L2010AA_S03_N403_POSTL_ZON_ZIP_CD],
[L2010AA_S04_REF02_ID_NR_OVL_EMPLYR_ID_NR],
'0' [L2000B_S01_HL04_HIERCHCL_CHILD_CD],
'P' [L2000B_S02_SBR01_PAYR_RESP_SEQNC_NR_CD],
'18' [L2000B_S02_SBR02_INDVDL_REL_CD],
'CI' [L2000B_S02_SBR09_CLM_FILNG_IND_CD]
```

Once the view is updated, we refresh the Chiapas Studio information to ensure that the information is being propagated:

```
[000001] ISA*00* *00* *ZZ*GP_CLINIC_ID *ZZ*FBN_HMO_ID
*140708*0121*^*00501*100000000*1*P*:
[000002] GS*HC*GP CLINIC_ID*FBN_HMO_ID*20140708*0121*1*X*005010X222A1
[000003] ST*837*100000000*005010X222A1
[000004] BHT*0019*00*996305*20140708*0121*CH
[000005] NM1*41*2*GEORGE PICKLE CLINIC*****46*999488801
[000006] PER*IC*GEORGE PICKLE JR*TE*4155551212
[000007] NM1*40*2*FLY BY NIGHT HMO*****46*999488802
[000008] HL*1**20*1
[000009] NM1*85*2*GEORGE PICKLE CLINIC*****XX*123400001
[000010] N3*4000 MAIN ST.*SUITE 201
[000011] N4*ANYTOWN*CA*000010000
[000012] REF*EI*999488801
[000013] HL*2*1*22*0
[000014] SBR*P*18*****CI
[000015] SE*13*100000000
[000016] GE*1*1
[000017] IEA*1*100000000
```

Note that we can observe the HL values being automatically generated, the SE segment count values automatically generated. Since we are not supplying these necessary values to the encoder, it generates these values on its own.

View:

vw_Export_FBN

Deliverable 1-E

The HIPAA Implementation Guide describes Hierarchical Level structures where information is aggregated by increasing specificity – one level is Subscriber, the next level is Claim, then Service Line.

Therefore, we need to rearrange the information within the CSV data source so that it fits in this hierarchical system. For Deliverable 1-E, we need to fulfill the requirements for the 2000B Hierarchical Level, which means supplying correct information for the 2010BA Subscriber Loop. We'll start with a simple aggregation SELECT statement:

```
SELECT      LAST_NAME,      FIRST_NAME,      REPLACE(SSN, '-', '') SSN,
            ADDRESS,        CITY,                STATE,
            ZIP_CODE,        REPLACE(BIRTH_DATE, '-', '') BIRTH_DATE,
            SEX
FROM        BIN_FBN
GROUP BY    LAST_NAME,      FIRST_NAME,      REPLACE(SSN, '-', ''),
            ADDRESS,        CITY,                STATE,
            ZIP_CODE,        REPLACE(BIRTH_DATE, '-', ''),
            SEX
```

Notice that by scrubbing out the dashes in the output, the fields now follow HIPAA-compliant syntax. Next, we map the results of this query into the View:

```
        SUB.LAST_NAME      [L2010BA_S01_NM103_SUB_LNAME_OVL_PERSN],
        SUB.FIRST_NAME     [L2010BA_S01_NM104_SUB_FNAME],
        SUB.SSN            [L2010BA_S01_NM109_PRIMRY_ID_OVL_MEM_ID_NR],
        SUB.ADDRESS        [L2010BA_S02_N301_ADRS_LIN],
        SUB.CITY           [L2010BA_S03_N401_CITY_NM],
        SUB.STATE          [L2010BA_S03_N402_STAT_CD],
        SUB.ZIP_CODE       [L2010BA_S03_N403_SUB_POSTL_ZON_ZIP_CD],
        SUB.BIRTH_DATE     [L2010BA_S04_DMG02_BIRTH_DT_OVL_DT_CCYYMMDD],
        SUB.SEX            [L2010BA_S04_DMG03_GENDR_CD]
FROM      PROD_TRAD_PARTNER_TP
INNER JOIN PROD_PROVIDER SENDR_PV ON TP.SENDR_PROV_ID = SENDR_PV.PROV_ID
INNER JOIN PROD_PROVIDER RECVR_PV ON TP.RECVR_PROV_ID = RECVR_PV.PROV_ID
INNER JOIN
(SELECT    DISTINCT
        LAST_NAME,      FIRST_NAME,      REPLACE(SSN, '-', '') SSN,
        ADDRESS,        CITY,                STATE,
        ZIP_CODE,        REPLACE(BIRTH_DATE, '-', '') BIRTH_DATE,
        SEX
FROM      BIN_FBN) SUB ON 1=1
```

Again, we alter the View and run the results through Chiapas Studio:

```
[000001] ISA*00*                *00*                *ZZ*GP_CLINIC_ID    *ZZ*FBN_HMO_ID
*140708*0936*^*00501*100000000*1*P*:
[000002] GS*HC*GP_CLINIC_ID*FBN_HMO_ID*20140708*0936*1*X*005010X222A1
[000003] ST*837*100000000*005010X222A1
[000004] BHT*0019*00*657746*20140708*0936*CH
[000005] NM1*41*2*GEORGE PICKLE CLINIC*****46*999488801
[000006] PER*IC*GEORGE PICKLE JR*TE*415551212
[000007] NM1*40*2*FLY BY NIGHT HMO*****46*999488802
[000008] HL*1**20*1
[000009] NM1*85*2*GEORGE PICKLE CLINIC*****XX*123400001
[000010] N3*4000 MAIN ST.*SUITE 201
[000011] N4*ANYTOWN*CA*000010000
[000012] REF*EI*999488801
[000013] HL*2*1*22*0
[000014] SBR*P*18*****CI
[000015] NM1*IL*1*KNIFE*MAC*MI*111229000
[000016] N3*123 WHITTLEBIRD ST
[000017] N4*ANYTOWN*CA*00001
[000018] DMG*D8*19730716*M
[000019] HL*3*1*22*0
[000020] SBR*P*18*****CI
[000021] NM1*IL*1*STEW*MARNEY***MI*111229999
[000022] N3*976 GINGERBREAD LN. APT 6
[000023] N4*ANYTOWN*CA*00001
```



```

[000024] DMG*D8*19620301*F
[000025] HL*4*1*22*0
[000026] SBR*P*18*****CI
[000027] NM1*IL*1*SUNDAE*DEVIN****MI*111228000
[000028] N3*842 TURBAN DR
[000029] N4*ANYTOWN*CA*00001
[000030] DMG*D8*19700114*M
[000031] SE*29*100000000
[000032] GE*1*1
[000033] IEA*1*100000000

```

This time, we see a more substantial (but still far from compliant or a valid EDI file) transaction, with the 2000B loops repeating once for each of the subscribers presented within the data file. This test validates we have achieved Deliverable 1-E.

View:

vw_Export_FBN

Deliverable 1-F

Within the raw EDI file, there are three distinct sets of Claim Header information across for data rows. We cannot count on there being one claim for every subscriber, so we will need to do a similar SELECT DISTINCT statement and then join it to the existing Subscriber extract. We add this clause to the end of the View syntax:

```

INNER JOIN
(SELECT DISTINCT      CLAIM_ID,      REPLACE(DX, '.', '') DX,
                      LAST_NAME,      FIRST_NAME,      REPLACE(SSN, '-', '') SSN
FROM      BIN_FBN
) CLM ON  SUB.LAST_NAME = CLM.LAST_NAME
AND      SUB.FIRST_NAME = CLM.FIRST_NAME
AND      SUB.SSN = CLM.SSN

```

Notice that we remove the dots from the Diagnosis Code and the dashes from the SSN – neither is accepted by HIPAA syntax conventions.

Extracting the data to the SELECT statement will be more difficult. Examining the existing vw_Export_837P loop, we see that there is a header-level summation of all claim charges. We will inherit the default values for the Claim Header that exist in the sample view, but we will match the four possible Service Line Charge fields to the CLAIM_ID and sum them up. We insert these additional lines to the VIEW statement prior to the FROM clause:

```

      CLM.CLAIM_ID      [L2300_S01_CLM01_PATNT_CONTRL_NR],
      (
        SELECT SUM(CONVERT(MONEY, SERVICE_CHARGE_1) +
                    CONVERT(MONEY, SERVICE_CHARGE_2) +
                    CONVERT(MONEY, SERVICE_CHARGE_3) +
                    CONVERT(MONEY, SERVICE_CHARGE_4))
        FROM      BIN_FBN
        WHERE     CLAIM_ID = CLM.CLAIM_ID
      )
      [L2300_S01_CLM02_TOTL_CLM_CHG_AMT],
      [L2300_S01_CLM05_01_PLAC_SVC_CD],
      [L2300_S01_CLM05_03_CLM_FREQNCY_CD_OVL_PLAC_SVC_CODS_PROF_DENTL],
      [L2300_S01_CLM06_PROV_SUPLR_SIGNTR_IND],
      [L2300_S01_CLM07_ASGNMNT_PLAN_PARTCPTN_CD],
      [L2300_S01_CLM08_BENFTS_ASGNMNT_CERT_IND],
      [L2300_S01_CLM09_RELS_NFO_CD],
      [L2300_S43_HI01_02_DIAG_CD_OVL_ICD9CM_PRINCPL_DIAG]
      CLM.DX

```

When we refresh the Chiapas Studio output, we immediately see a problem with the output:

```
[000035] CLM*8980153156*69.0000***11:B:1*Y*A*Y*Y
[000036] HI*BK:27900
```

The extended zeroes here at the end of the SUM statement are not HIPAA compliant – therefore, we can use the fx_HIPAA_NUMBER function built-in to the chiapasData database to put this value into a HIPAA compliant format:

```
(      SELECT      dbo.fx_HIPAA_NUMBER
                (SUM(CONVERT(MONEY, SERVICE_CHARGE_1) +
                    CONVERT(MONEY, SERVICE_CHARGE_2) +
                    CONVERT(MONEY, SERVICE_CHARGE_3) +
                    CONVERT(MONEY, SERVICE_CHARGE_4)))
      FROM      BIN_FBN
      WHERE     CLAIM_ID = CLM.CLAIM_ID
      )
      [L2300_S01_CLM02_TOTL_CLM_CHG_AMT],
```

After altering the vw_Extract_FBN view again with the updated syntax and refreshing in Chiapas Studio, the extra zeroes are gone and the sums are in a compliant syntax:

```
[000027] CLM*8980153155*205***11:B:1*Y*A*Y*Y
[000028] HI*BK:30289
```

View:

vw_Export_FBN

Deliverable 1-G

Now, we are almost finished with this ETL extract to convert the data into HIPAA compliant syntax. For the final step, we need to split the four Service Line groups so that they appear as separate rows on the extract. Adding after the CLM clause:

```
INNER JOIN
( SELECT CLAIM_ID, SERVICE_CODE_1 CD, SERVICE_DATE_1 DT, SERVICE_CHARGE_1 CH
  FROM  BIN_FBN UNION
  SELECT CLAIM_ID, SERVICE_CODE_2 CD, SERVICE_DATE_2 DT, SERVICE_CHARGE_2 CH
  FROM  BIN_FBN UNION
  SELECT CLAIM_ID, SERVICE_CODE_3 CD, SERVICE_DATE_3 DT, SERVICE_CHARGE_3 CH
  FROM  BIN_FBN UNION
  SELECT CLAIM_ID, SERVICE_CODE_4 CD, SERVICE_DATE_4 DT, SERVICE_CHARGE_4 CH
  FROM  BIN_FBN )
SVC ON SVC.CLAIM_ID = CLM.CLAIM_ID AND SVC.CD <> ''
```

This logic will ensure that the Service Lines will be rearranged to a row-by-row fashion. Now, for the values in the SELECT clause after the CLM.DX:

```
SVC.CD      [L2400_S02_SV101_02_PROC_CD_OVL_HCPCS_CD],
dbo.fx_HIPAA_NUMBER(SVC.CH)
            [L2400_S02_SV102_LIN_ITM_CHG_AMT],
'1'         [L2400_S02_SV104_SVC_UNT_CT_OVL_UNT],
'1'         [L2400_S02_SV107_01_DIAG_CD_POINTR],
REPLACE(SVC.DT, '-', '')
            [L2400_S11_DTP03_SVC_DT_OVL_SVC_D8]
```

Next, we alter the View and refresh the segment output in Chiapas Studio. Everything looks good:

```

[000035] HL*4*1*22*0
[000036] SBR*P*18*****CI
[000037] NM1*IL*1*SUNDAE*DEVIN****MI*111228000
[000038] N3*842 TURBAN DR
[000039] N4*ANYTOWN*CA*00001
[000040] DMG*D8*19700114*M
[000041] CLM*8980153156*69***11:B:1*Y*A*Y*Y
[000042] HI*BK:27900
[000043] LX*1
[000044] SV1*HC:80048*15*UN*1***1
[000045] DTP*472*D8*20140702
[000046] LX*2
[000047] SV1*HC:80076*15*UN*1***1
[000048] DTP*472*D8*20140702
[000049] LX*3
[000050] SV1*HC:80202*17*UN*1***1
[000051] DTP*472*D8*20140702
[000052] LX*4
[000053] SV1*HC:85025*10*UN*1***1
[000054] DTP*472*D8*20140702
[000055] LX*5
[000056] SV1*HC:85652*12*UN*1***1

```

At this point, we have what appears to be a HIPAA compliant file. Because Chiapas EDI Enterprise provides SNIP Level 1 and 2 compliance checking, however, we do not need to take this on faith – we just press the DECODE button and see the results:

```

[000017] N4*ANYTOWN*CA*00001
[000018] DMG*D8*19620301*F
[000019] CLM*8980153154*42***11:B:1*Y*A*Y*Y
Mandatory Loop 2010BB Not Found | | ELE00
[000020] HI*BK:V7283
[000021] LX*1
[000022] SV1*HC:99213*42*UN*1***1

```

When we research Loop 2010BB in the HIPAA Implementation Guide, we see that the Payer loop is required. In this case, the Payer Provider is the same as the Receiving Provider, so we merely have to repeat some mappings for the 2010BB loop and we fulfill the requirement. Here is the vw_Extract_FBN in its entirety:

```

ALTER VIEW [dbo].[vw_Export_FBN]
AS
SELECT  ISA02                [X1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO],
        ISA04                [OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO],
        ISA06_SENDR_ID       [OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID],
        ISA08_RECVR_ID       [OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID],
        ISA11_REPTN_SEP      [OEISA_S01_ISA11_REPTN_SEPRTR],
        '00501'              [OEISA_S01_ISA12_VERSN_NR],
        ICN                  [OEISA_S01_ISA13_ICN],
        ISA14_ACK_REQ        [OEISA_S01_ISA14_ACK_REQ],
        ISA15_USG_IND        [OEISA_S01_ISA15_USG_IND],
        ISA16_SUBELE_SEP     [OEISA_S01_ISA16_SUBELE_SEP],
        ICN                  [OEISA_S03_IEA02_ICN],
        GS02_APP_SENDR_CD    [GSHDR_S01_GS02_APP_SENDR_CD],
        GS03_APP_RECVR_CD    [GSHDR_S01_GS03_APP_RECVR_CD],
        GCN                  [GSHDR_S01_GS06_GCN],
        '005010X222A1'      [GSHDR_S01_GS08_VERS_ID_CD],
        GCN                  [GSHDR_S03_GE02_GCN],
        TCN                  [STHDR_S01_ST02_CONTRL_NR],
        '005010X222A1'      [STHDR_S01_ST03_IMPLMNTN_GUID_VERSN_NM],
        TCN                  [STHDR_S06_SE02_TCN],
        '0019'              [STHDR_S02_BHT01_STRUCTR_CD],
        '00'                [STHDR_S02_BHT02_TS_PURPS_CD],

```

```

RIGHT(CONVERT(VARCHAR,RAND( (DATEPART(mm, GETDATE()) * 100000)+
(DATEPART(ss, GETDATE()) * 1000 )+DATEPART(ms, GETDATE()) ),6)
[STHDR_S02_BHT03_ORGNTR_APLCTN_TRANSCTN_ID],
'CH' [STHDR_S02_BHT06_CLM_ENCNTR_ID],
SENDER_PV.COMPANY_NM
[L1000A_S01_NM103_SUBMTR_LAST_ORG_NM_OVL_NONPRSN_ENTY],
SENDER_PV.TAX_ID [L1000A_S01_NM109_SUBMTR_ID_OVL_ETN_NR],
SENDER_PV.CONTACT_NM [L1000A_S02_PER02_CONTACT_NM],
SENDER_PV.WORK_PH [L1000A_S02_PER04_COMM_NR_OVL_TELPHN],
RECEIVER_PV.COMPANY_NM [L1000B_S01_NM103_RECEIVER_NM_OVL_NONPRSN_ENTY],
RECEIVER_PV.TAX_ID [L1000B_S01_NM109_PRIMARY_ID_OVL_ETN_NR],
'1' [L2000A_S01_HL04_HIERCHCL_CHILD_CD],
SENDER_PV.COMPANY_NM
[L2010AA_S01_NM103_BILNG_PROV_ORGL_NM_OVL_NONPRSN_ENTY],
SENDER_PV.NPI_ID
[L2010AA_S01_NM109_PROV_ID_OVL_CENTRS_MEDCR_MEDCD_SVCS],
SENDER_PV.ADD1 [L2010AA_S02_N301_BILNG_PROV_ADRS_LIN],
SENDER_PV.ADD2 [L2010AA_S02_N302_BILNG_PROV_ADRS_LIN],
SENDER_PV.CITY [L2010AA_S03_N401_BILNG_PROV_CITY_NM],
SENDER_PV.STATE [L2010AA_S03_N402_STAT_PROVNC_CD],
CONVERT(VARCHAR,SENDER_PV.ZIP_CD) + '0000'
[L2010AA_S03_N403_POSTL_ZON_ZIP_CD],
SENDER_PV.TAX_ID [L2010AA_S04_REF02_ID_NR_OVL_EMPLOYR_ID_NR],
'0' [L2000B_S01_HL04_HIERCHCL_CHILD_CD],
'P' [L2000B_S02_SBR01_PAYR_RESP_SEQNC_NR_CD],
'18' [L2000B_S02_SBR02_INVDL_REL_CD],
'CI' [L2000B_S02_SBR09_CLM_FILNG_IND_CD],
SUB.LAST_NAME [L2010BA_S01_NM103_SUB_LNAME_OVL_PERSN],
SUB.FIRST_NAME [L2010BA_S01_NM104_SUB_FNAME],
SUB.SSN [L2010BA_S01_NM109_PRIMARY_ID_OVL_MEM_ID_NR],
SUB.ADDRESS [L2010BA_S02_N301_ADRS_LIN],
SUB.CITY [L2010BA_S03_N401_CITY_NM],
SUB.STATE [L2010BA_S03_N402_STAT_CD],
SUB.ZIP_CODE [L2010BA_S03_N403_SUB_POSTL_ZON_ZIP_CD],
SUB.BIRTH_DATE [L2010BA_S04_DM02_BIRTH_DT_OVL_DT_CCYYMMDD],
SUB.SEX [L2010BA_S04_DM03_GENDR_CD],
RECEIVER_PV.COMPANY_NM [L2010BB_S01_NM103_PAYR_NM_OVL_NONPRSN_ENTY],
RECEIVER_PV.TAX_ID [L2010BB_S01_NM109_PAYR_ID_OVL_PAYR_ID],
RECEIVER_PV.ADD1 [L2010BB_S02_N301_ADRS_LIN],
RECEIVER_PV.CITY [L2010BB_S03_N401_CITY_NM],
RECEIVER_PV.STATE [L2010BB_S03_N402_PAYR_STAT_PROVNC_CD],
CONVERT(VARCHAR,RECEIVER_PV.ZIP_CD) + '0000'
[L2010BB_S03_N403_PAYR_POSTL_ZON_ZIP_CD],
CLM.CLAIM_ID [L2300_S01_CLM01_PATNT_CTRLNR],
(SELECT dbo.fx_HIPAA_NUMBER
(SUM( CONVERT(MONEY, SERVICE_CHARGE_1 +
COALESCE(SERVICE_CHARGE_2, CONVERT(MONEY,0.00)) +
COALESCE(SERVICE_CHARGE_3, CONVERT(MONEY,0.00)) +
COALESCE(SERVICE_CHARGE_4, CONVERT(MONEY,0.00))))
FROM BIN_FBN
WHERE CLAIM_ID = CLM.CLAIM_ID)
[L2300_S01_CLM02_TOTL_CLM_CHG_AMT],
'11' [L2300_S01_CLM05_01_PLAC_SVC_CD],
'1' [L2300_S01_CLM05_03_CLM_FREQNCY_CD_OVL_PLAC_SVC_CODS_PROF_DENTL],
'Y' [L2300_S01_CLM06_PROV_SUPLR_SIGNTR_IND],
'A' [L2300_S01_CLM07_ASGNMNT_PLAN_PARTCPTN_CD],
'Y' [L2300_S01_CLM08_BENFTS_ASGNMNT_CERT_IND],
'Y' [L2300_S01_CLM09_RELS_NFO_CD],
CLM.DX [L2300_S43_HI01_02_DIAG_CD_OVL_ICD9CM_PRINCPL_DIAG],
SVC.CD [L2400_S02_SV101_02_PROC_CD_OVL_HCPCS_CD],
dbo.fx_HIPAA_NUMBER(SVC.CH)
[L2400_S02_SV102_LIN_ITM_CHG_AMT],
'1' [L2400_S02_SV104_SVC_UNT_CT_OVL_UNT],
'1' [L2400_S02_SV107_01_DIAG_CD_POINTR],
REPLACE(SVC.DT, '-', ''')
[L2400_S11_DTP03_SVC_DT_OVL_SVC_D8]
FROM PROD_TRAD_PARTNER_TP
INNER JOIN PROD_PROVIDER SENDER_PV ON TP.SENDER_PROV_ID = SENDER_PV.PROV_ID
INNER JOIN PROD_PROVIDER RECEIVER_PV ON TP.RECEIVER_PROV_ID = RECEIVER_PV.PROV_ID

```

```

INNER JOIN
(
SELECT      DISTINCT
      LAST_NAME,      FIRST_NAME,      REPLACE(SSN, '-', '') SSN,
      ADDRESS,      CITY,      STATE,
      ZIP_CODE,      REPLACE(BIRTH_DATE, '-', '') BIRTH_DATE,
      SEX
FROM      BIN_FBN
) SUB ON 1=1
INNER JOIN
(SELECT      DISTINCT      CLAIM_ID,      REPLACE(DX, '.', '') DX,
      LAST_NAME,      FIRST_NAME,      REPLACE(SSN, '-', '') SSN
FROM      BIN_FBN
) CLM ON  SUB.LAST_NAME = CLM.LAST_NAME
AND      SUB.FIRST_NAME = CLM.FIRST_NAME
AND      SUB.SSN = CLM.SSN
INNER JOIN
(
SELECT CLAIM_ID, SERVICE_CODE_1 CD, SERVICE_DATE_1 DT, SERVICE_CHARGE_1 CH
FROM BIN_FBN UNION
SELECT CLAIM_ID, SERVICE_CODE_2 CD, SERVICE_DATE_2 DT, SERVICE_CHARGE_2 CH
FROM BIN_FBN UNION
SELECT CLAIM_ID, SERVICE_CODE_3 CD, SERVICE_DATE_3 DT, SERVICE_CHARGE_3 CH
FROM BIN_FBN UNION
SELECT CLAIM_ID, SERVICE_CODE_4 CD, SERVICE_DATE_4 DT, SERVICE_CHARGE_4 CH
FROM BIN_FBN )
SVC ON SVC.CLAIM_ID = CLM.CLAIM_ID AND SVC.CD <> ''

```

Deliverable 2

As described in the assessment, this is where we tie in the ETL we established in the previously deliverable into a fully automated solution. This section walks us through these deliverables one step at a time.

Deliverable 2-A

Deliverable 2-B

The first thing we must do is set up the automation infrastructure necessary to process the incoming legacy file into a new EDI file.

First, we create the folders off the of 'pipeline' directory on the C: drive:

```
cd c:\chiapas\pipeline
mkdir 013_CSVLegacy
cd 013_CSVLegacy
mkdir 01_in_csv
mkdir 02_in_csv
mkdir 03_out_edi
```

Next, we set up a LOCAL UPLOAD trigger that will pull new CSV files from the 01 directory to the 02 directory, and then process a workflow on that file. We open Chiapas Studio and select the pulldown Window -> Scheduling Interface, then press the NEW button to create a new trigger. Selecting the newly created trigger in the 'All Triggers' tab, we enter in a name of 'Legacy CSV Process' and and press SAVE. Then, we go to Polling Info, and enter in the following information:

Local Upload File Poll	
Local UL Dir	C:\chiapas\pipeline\013_CSVLegacy\01_in_csv
Dest Dir	C:\chiapas\pipeline\013_CSVLegacy\02_in_csv

Then we press SAVE. We cannot enable the automation until we have the Workflow set up – and prior to that, we need the ability to import the data from the legacy CSV text file into a database.

The following Stored Procedure does the following. Accepting the EVENT_ID and a filename, it copies the data in the CSV file and creates a new BIN_LOG entry. It should be created on the chiapasData database.

```
CREATE PROCEDURE USP_PROCESS_CSV(@EVENT_ID INT,
                                  @FN VARCHAR(200))
AS
BEGIN
    -- Clone the BIN_FBN table to a new temp table
    DECLARE @SQL VARCHAR(MAX), @TMP_ID INT
    SET @SQL = 'SELECT TOP 0 * INTO TEMP' +
        CONVERT(VARCHAR, @EVENT_ID) +
        ' FROM BIN_FBN; ALTER TABLE TEMP' +
        CONVERT(VARCHAR, @EVENT_ID) +
        ' DROP COLUMN BIN_ID;'
    EXEC(@SQL)
    -- Bulk Insert the CSV legacy data
    SET @SQL = 'BULK INSERT TEMP' +
        CONVERT(VARCHAR, @EVENT_ID) +
        ' FROM ''' + @FN +
        ''' WITH (FIELDTERMINATOR = ''',''', '' ' +
        'ROWTERMINATOR = '''\n'');UPDATE TEMP' +
```

```

        CONVERT(VARCHAR, @EVENT_ID) +
        ' SET FILE_DATE = REPLACE(FILE_DATE, '''', ''');'
EXEC(@SQL)
-- Create new BIN_LOG entry and insert the data into
-- the BIN_FBN table
BEGIN TRAN
    INSERT INTO BIN_LOG
    ( BIN_FILENAME,
      BIN_RECV_DATE,
      BIN_TYPE,
      EVENT_ID
    ) SELECT @FN,
      GETDATE(),
      'BIN_FBN',
      @EVENT_ID

    SET @TMP_ID = SCOPE_IDENTITY()
COMMIT TRAN
SET @SQL = 'INSERT INTO BIN_FBN SELECT ' +
  CONVERT(VARCHAR, @TMP_ID) + ', * FROM TEMP' +
  CONVERT(VARCHAR, @EVENT_ID) + ';DROP TABLE TEMP' +
  CONVERT(VARCHAR, @EVENT_ID)
EXEC(@SQL)
END

```

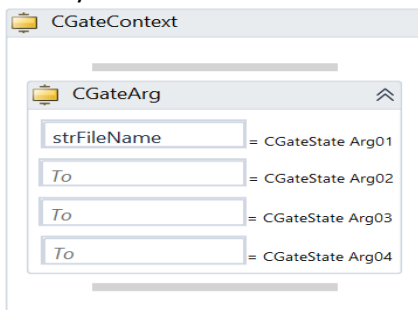
Now, to tie everything together. We've created a trigger already, but without a Workflow, it has no function. Open Chiapas Studio and switch to the Workflow Interface, and create a new workflow.

The Stored Procedure we've just created is expecting both a file name and an EVENT_ID. The filename is passed as the first argument within the automation. The EVENT_ID is available via the CGateQueryState activity. However, we will need two string variables to hold these values – so to start with, we click on the CGateContext, click Variables, and add these two values:

Name	Variable type
strEventID	String
strFileName	String

Create Variable

Next, we populate the variables we need in order to execute the stored procedure. The CGateArg activity will load the automation filename into the strFileName variable:



Next, we put in a CGateQueryState activity and fill in the Properties:

Misc	
DisplayName	CGateQueryState
Query	"EVENT_ID" ...
Result	strEventID ...

Finally, we add in CGateCNNSQLExec activity and fill in the Properties. Because we created the Stored Procedure on the chiapasData database that is part of the automation environment, we do not need to open up a new SQL Connection – we reference this ‘always on’ connection with the name “TriggerDB”:

CNNName:
"TriggerDB"

CNNSQL:
"EXEC USP_PROCESS_CSV " + strEventID + ",'" + strFileName + "'"

Then, we save the whole thing as “CSVLegacy.xml” to the C:\chiapas\xaml directory. Finally, we return to the Scheduler Interface and set the Workflow to the newly created workflow file:

C:\chiapas\xaml\CSVLegacy.xml

Then, click on ‘Enabled’ and SAVE.

To test, we simply drag our sample CSV file into the 01_in_csv directory. It should disappear, and then we can validate the BIN_LOG and BIN_FBN rows:

```
SELECT * FROM BIN_LOG
SELECT * FROM BIN_FBN
```

This validates that Deliverables 2-A and 2-B are fulfilled.

Deliverable 2-C

The automation infrastructure bundled with Chiapas EDI is focused on making certain standard operations fairly straightforward, including the task of exporting an EDI file present within the BIN tables. In order to re-use this functionality, we will need to create a new BIN_ID and use the vw_Export_FBN view we created earlier to populate the BIN_837P table.

Earlier, we used the vw_Export_FBN to test our mappings as a direct input to Chiapas EDI. As it does not link to the BIN_ID, it is only capable of returning all of the billing information in the BIN_FBN table at once and is therefore more of a testing resource than a production-ready component. This SQL logic can be used as an INSERT statement into the BIN_837P with some minor modifications, however.

First, we’ll need to make some alterations to the receiving BIN_837P table to ensure it can receive all of the fields:

```
ALTER TABLE BIN_837P ADD [STHDR_S01_ST03_IMPLMNTN_GUID_VERSN_NM] VARCHAR(200)
ALTER TABLE BIN_837P ADD [L2000A_S01_HL04_HIERCHCL_CHILD_CD] VARCHAR(200)
ALTER TABLE BIN_837P ADD [L2010AA_S01_NM103_BILNG_PROV_ORGL_NM_OVL_NONPRSN_ENTY] VARCHAR(200)
ALTER TABLE BIN_837P ADD [L2010AA_S01_NM109_PROV_ID_OVL_CENTRS_MEDCR_MEDCD_SVCS] VARCHAR(200)
```



```
ALTER TABLE BIN_837P ADD [L2010AA_S02_N302_BILNG_PROV_ADRS_LIN] VARCHAR(200)
```

Go ahead and ALTER the USP_PROCESS_CSV stored procedure, and add these lines to the end:

```
-- Create a new BIN_LOG entry for the 837P data we're
-- about to create. Set the filename to be a current timestamp:
DECLARE @TMP_ID2 INT
BEGIN TRAN
    INSERT INTO BIN_LOG
    ( BIN_FILENAME,
      BIN_RECV_DATE,
      BIN_TYPE,
      EVENT_ID
    ) SELECT @FN,
      GETDATE(),
      '5010_837P_A1',
      @EVENT_ID

    SET @TMP_ID2 = SCOPE_IDENTITY()
COMMIT TRAN

INSERT INTO BIN_837P
(BIN_ID
,[X1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO]
,[OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO]
,[OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID]
,[OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID]
,[OEISA_S01_ISA11_REPTN_SEPRTR]
,[OEISA_S01_ISA12_VERSN_NR]
,[OEISA_S01_ISA13_ICN]
,[OEISA_S01_ISA14_ACK_REQ]
,[OEISA_S01_ISA15_USG_IND]
,[OEISA_S01_ISA16_SUBELE_SEP]
,[OEISA_S03_IEA02_ICN]
,[GSHDR_S01_GS02_APP_SENDR_CD]
,[GSHDR_S01_GS03_APP_RECVR_CD]
,[GSHDR_S01_GS06_GCN]
,[GSHDR_S01_GS08_VERS_ID_CD]
,[GSHDR_S03_GE02_GCN]
,[STHDR_S01_ST02_CONTRL_NR]
,[STHDR_S01_ST03_IMPLMNTN_GUID_VERSN_NM]
,[STHDR_S06_SE02_TCN]
,[STHDR_S02_BHT01_STRUCTR_CD]
,[STHDR_S02_BHT02_TS_PURPS_CD]
,[STHDR_S02_BHT03_ORGNTR_APLCTN_TRANSCTN_ID]
,[STHDR_S02_BHT06_CLM_ENCINTR_ID]
,[L1000A_S01_NM103_SUBMTR_LAST_ORG_NM_OVL_NONPRSN_ENTY]
,[L1000A_S01_NM109_SUBMTR_ID_OVL_ETN_NR]
,[L1000A_S02_PER02_CONTCNT_NM]
,[L1000A_S02_PER04_COMM_NR_OVL_TELPHN]
,[L1000B_S01_NM103_RECVR_NM_OVL_NONPRSN_ENTY]
,[L1000B_S01_NM109_PRIMARY_ID_OVL_ETN_NR]
,[L2000A_S01_HL04_HIERCHCL_CHILD_CD]
,[L2010AA_S01_NM103_BILNG_PROV_ORGL_NM_OVL_NONPRSN_ENTY]
,[L2010AA_S01_NM109_PROV_ID_OVL_CENTRS_MEDCR_MEDCD_SVCS]
,[L2010AA_S02_N301_BILNG_PROV_ADRS_LIN]
,[L2010AA_S02_N302_BILNG_PROV_ADRS_LIN]
,[L2010AA_S03_N401_BILNG_PROV_CITY_NM]
,[L2010AA_S03_N402_STAT_PROVNC_CD]
,[L2010AA_S03_N403_POSTL_ZON_ZIP_CD]
,[L2010AA_S04_REF02_ID_NR_OVL_EMPTYR_ID_NR]
,[L2000B_S01_HL04_HIERCHCL_CHILD_CD]
,[L2000B_S02_SBR01_PAYR_RESP_SEQNC_NR_CD]
,[L2000B_S02_SBR02_INVDL_REL_CD]
,[L2000B_S02_SBR09_CLM_FILNG_IND_CD]
,[L2010BA_S01_NM103_SUB_LNAME_OVL_PERSN]
,[L2010BA_S01_NM104_SUB_FNAME])
```

```

, [L2010BA_S01_NM109_PRIMARY_ID_OVL_MEM_ID_NR]
, [L2010BA_S02_N301_ADRS_LIN]
, [L2010BA_S03_N401_CITY_NM]
, [L2010BA_S03_N402_STAT_CD]
, [L2010BA_S03_N403_SUB_POSTL_ZON_ZIP_CD]
, [L2010BA_S04_DMG02_BIRTH_DT_OVL_DT_CCYMMDD]
, [L2010BA_S04_DMG03_GENDR_CD]
, [L2010BB_S01_NM103_PAYR_NM_OVL_NONPRSN_ENTY]
, [L2010BB_S01_NM109_PAYR_ID_OVL_PAYR_ID]
, [L2010BB_S02_N301_ADRS_LIN]
, [L2010BB_S03_N401_CITY_NM]
, [L2010BB_S03_N402_PAYR_STAT_PROVNC_CD]
, [L2010BB_S03_N403_PAYR_POSTL_ZON_ZIP_CD]
, [L2300_S01_CLM01_PATNT_CONTRL_NR]
, [L2300_S01_CLM02_TOTL_CLM_CHG_AMT]
, [L2300_S01_CLM05_01_PLAC_SVC_CD]
, [L2300_S01_CLM05_03_CLM_FREQNCY_CD_OVL_PLAC_SVC_CODS_PROF_DENTL]
, [L2300_S01_CLM06_PROV_SUPLR_SIGNTR_IND]
, [L2300_S01_CLM07_ASGNMNT_PLAN_PARTCPTN_CD]
, [L2300_S01_CLM08_BENFTS_ASGNMNT_CERT_IND]
, [L2300_S01_CLM09_RELS_NFO_CD]
, [L2300_S43_HI01_02_DIAG_CD_OVL_ICD9CM_PRINCPL_DIAG]
, [L2400_S02_SV101_02_PROC_CD_OVL_HCPCS_CD]
, [L2400_S02_SV102_LIN_ITM_CHG_AMT]
, [L2400_S02_SV104_SVC_UNT_CT_OVL_UNT]
, [L2400_S02_SV107_01_DIAG_CD_POINTR]
, [L2400_S11_DTP03_SVC_DT_OVL_SVC_D8])
SELECT @TMP_ID2,
        ISA02           [X1_OEISA_S01_ISA02_AUTH_NFO_OVL_NO_AUTH_NFO],
        ISA04           [OEISA_S01_ISA04_SEC_NFO_OVL_NO_SEC_NFO],
        ISA06_SENDR_ID  [OEISA_S01_ISA06_SENDR_ID_OVL_MUTLY_DEFND_ID],
        ISA08_RECVR_ID  [OEISA_S01_ISA08_RECVR_ID_OVL_MUTLY_DEFND_ID],
        ISA11_REPTN_SEP [OEISA_S01_ISA11_REPTN_SEPRTR],
        '00501'         [OEISA_S01_ISA12_VERSN_NR],
        ICN             [OEISA_S01_ISA13_ICN],
        ISA14_ACK_REQ   [OEISA_S01_ISA14_ACK_REQ],
        ISA15_USG_IND   [OEISA_S01_ISA15_USG_IND],
        ISA16_SUBELE_SEP [OEISA_S01_ISA16_SUBELE_SEP],
        ICN             [OEISA_S03_IEA02_ICN],
        GS02_APP_SENDR_CD [GSHDR_S01_GS02_APP_SENDR_CD],
        GS03_APP_RECVR_CD [GSHDR_S01_GS03_APP_RECVR_CD],
        GCN             [GSHDR_S01_GS06_GCN],
        '005010X222A1'  [GSHDR_S01_GS08_VERS_ID_CD],
        GCN             [GSHDR_S03_GE02_GCN],
        TCN             [STHDR_S01_ST02_CONTRL_NR],
        '005010X222A1'  [STHDR_S01_ST03_IMPLMNTN_GUID_VERSN_NM],
        TCN             [STHDR_S06_SE02_TCN],
        '0019'          [STHDR_S02_BHT01_STRUCTR_CD],
        '00'            [STHDR_S02_BHT02_TS_PURPS_CD],
        RIGHT(CONVERT(VARCHAR, RAND( (DATEPART(mm, GETDATE()) * 100000)+
        (DATEPART(ss, GETDATE()) * 1000 )+DATEPART(ms, GETDATE()) )),6)
        [STHDR_S02_BHT03_ORGNTR_APLCTN_TRANSCTN_ID],
        'CH'            [STHDR_S02_BHT06_CLM_ENCNR_ID],
        SENDR_PV.COMPANY_NM
        [L1000A_S01_NM103_SUBMTR_LAST_ORG_NM_OVL_NONPRSN_ENTY],
        SENDR_PV.TAX_ID   [L1000A_S01_NM109_SUBMTR_ID_OVL_ETN_NR],
        SENDR_PV.CONTACT_NM [L1000A_S02_PER02_CONTCNT_NM],
        SENDR_PV.WORK_PH   [L1000A_S02_PER04_COMM_NR_OVL_TELPHN],
        RECVR_PV.COMPANY_NM [L1000B_S01_NM103_RECVR_NM_OVL_NONPRSN_ENTY],
        RECVR_PV.TAX_ID   [L1000B_S01_NM109_PRIMARY_ID_OVL_ETN_NR],
        '1'              [L2000A_S01_HL04_HIERCHCL_CHILD_CD],
        SENDR_PV.COMPANY_NM
        [L2010AA_S01_NM103_BILNG_PROV_ORGL_NM_OVL_NONPRSN_ENTY],
        SENDR_PV.NPI_ID
        [L2010AA_S01_NM109_PROV_ID_OVL_CENTRS_MEDCR_MEDCD_SVCS],
        SENDR_PV.ADD1     [L2010AA_S02_N301_BILNG_PROV_ADRS_LIN],
        SENDR_PV.ADD2     [L2010AA_S02_N302_BILNG_PROV_ADRS_LIN],
        SENDR_PV.CITY     [L2010AA_S03_N401_BILNG_PROV_CITY_NM],
        SENDR_PV.STATE    [L2010AA_S03_N402_STAT_PROVNC_CD],

```

```

        CONVERT(VARCHAR,SENR_PV.ZIP_CD) + '0000'
        [L2010AA_S03_N403_POSTL_ZON_ZIP_CD],
SENR_PV.TAX_ID      [L2010AA_S04_REF02_ID_NR_OVL_EMPLYR_ID_NR],
'0'                [L2000B_S01_HL04_HIERCHCL_CHILD_CD],
'P'                [L2000B_S02_SBR01_PAYR_RESP_SEQNC_NR_CD],
'18'               [L2000B_S02_SBR02_INDVDL_REL_CD],
'CI'               [L2000B_S02_SBR09_CLM_FILNG_IND_CD],
SUB.LAST_NAME      [L2010BA_S01_NM103_SUB_LNAME_OVL_PERSN],
SUB.FIRST_NAME     [L2010BA_S01_NM104_SUB_FNAME],
SUB.SSN            [L2010BA_S01_NM109_PRIMARY_ID_OVL_MEM_ID_NR],
SUB.ADDRESS        [L2010BA_S02_N301_ADRS_LIN],
SUB.CITY           [L2010BA_S03_N401_CITY_NM],
SUB.STATE          [L2010BA_S03_N402_STAT_CD],
SUB.ZIP_CODE       [L2010BA_S03_N403_SUB_POSTL_ZON_ZIP_CD],
SUB.BIRTH_DATE     [L2010BA_S04_DM02_BIRTH_DT_OVL_DT_CCYMMDD],
SUB.SEX            [L2010BA_S04_DM03_GENDR_CD],
RECV_PV.COMPANY_NM [L2010BB_S01_NM103_PAYR_NM_OVL_NONPRSN_ENTY],
RECV_PV.TAX_ID     [L2010BB_S01_NM109_PAYR_ID_OVL_PAYR_ID],
RECV_PV.ADD1       [L2010BB_S02_N301_ADRS_LIN],
RECV_PV.CITY       [L2010BB_S03_N401_CITY_NM],
RECV_PV.STATE      [L2010BB_S03_N402_PAYR_STAT_PROVNC_CD],
CONVERT(VARCHAR,RECV_PV.ZIP_CD) + '0000'
        [L2010BB_S03_N403_PAYR_POSTL_ZON_ZIP_CD],
CLM.CLAIM_ID       [L2300_S01_CLM01_PATNT_CONTRL_NR],
(
    SELECT dbo.fx_HIPAA_NUMBER
        (SUM(
            CONVERT(MONEY, SERVICE_CHARGE_1 +
            COALESCE(SERVICE_CHARGE_2, CONVERT(MONEY,0.00)) +
            COALESCE(SERVICE_CHARGE_3, CONVERT(MONEY,0.00)) +
            COALESCE(SERVICE_CHARGE_4, CONVERT(MONEY,0.00))))
    FROM BIN_FBN
    WHERE CLAIM_ID = CLM.CLAIM_ID AND BIN_ID = @TMP_ID )
    [L2300_S01_CLM02_TOTL_CLM_CHG_AMT],
'11'              [L2300_S01_CLM05_01_PLAC_SVC_CD],
'1'                [L2300_S01_CLM05_03_CLM_FREQNCY_CD_OVL_PLAC_SVC_CODS_PROF_DENTL],
'Y'                [L2300_S01_CLM06_PROV_SUPLR_SIGNTR_IND],
'A'                [L2300_S01_CLM07_ASGNMNT_PLAN_PARTCPTN_CD],
'Y'                [L2300_S01_CLM08_BENFTS_ASGNMNT_CERT_IND],
'Y'                [L2300_S01_CLM09_RELS_NFO_CD],
CLM.DX             [L2300_S43_HI01_02_DIAG_CD_OVL_ICD9CM_PRINCPL_DIAG],
SVC.CD             [L2400_S02_SV101_02_PROC_CD_OVL_HCPCS_CD],
dbo.fx_HIPAA_NUMBER(SVC.CH)
    [L2400_S02_SV102_LIN_ITM_CHG_AMT],
'1'                [L2400_S02_SV104_SVC_UNT_CT_OVL_UNT],
'1'                [L2400_S02_SV107_01_DIAG_CD_POINTR],
REPLACE(SVC.DT, '-', '')
    [L2400_S11_DTP03_SVC_DT_OVL_SVC_D8]
FROM PROD_TRAD_PARTNER TP
INNER JOIN PROD_PROVIDER SENR_PV ON TP.SENR_PROV_ID = SENR_PV.PROV_ID
INNER JOIN PROD_PROVIDER RECV_PV ON TP.RECV_PROV_ID = RECV_PV.PROV_ID
INNER JOIN
(
    SELECT DISTINCT
        LAST_NAME,      FIRST_NAME,      REPLACE(SSN, '-', '') SSN,
        ADDRESS,        CITY,            STATE,
        ZIP_CODE,       REPLACE(BIRTH_DATE, '-', '') BIRTH_DATE,
        SEX
    FROM BIN_FBN WHERE BIN_ID = @TMP_ID
) SUB ON 1=1
INNER JOIN
(
    SELECT DISTINCT CLAIM_ID,      REPLACE(DX, '.', '') DX,
        LAST_NAME,      FIRST_NAME,  REPLACE(SSN, '-', '') SSN
    FROM BIN_FBN WHERE BIN_ID = @TMP_ID
) CLM ON SUB.LAST_NAME = CLM.LAST_NAME
AND SUB.FIRST_NAME = CLM.FIRST_NAME
AND SUB.SSN = CLM.SSN
INNER JOIN
(
    SELECT CLAIM_ID, SERVICE_CODE_1 CD, SERVICE_DATE_1 DT, SERVICE_CHARGE_1 CH
    FROM BIN_FBN WHERE BIN_ID = @TMP_ID UNION
    SELECT CLAIM_ID, SERVICE_CODE_2 CD, SERVICE_DATE_2 DT, SERVICE_CHARGE_2 CH

```

```

FROM BIN_FBN WHERE BIN_ID = @TMP_ID UNION
SELECT CLAIM_ID, SERVICE_CODE_3 CD, SERVICE_DATE_3 DT, SERVICE_CHARGE_3 CH
FROM BIN_FBN WHERE BIN_ID = @TMP_ID UNION
SELECT CLAIM_ID, SERVICE_CODE_4 CD, SERVICE_DATE_4 DT, SERVICE_CHARGE_4 CH
FROM BIN_FBN WHERE BIN_ID = @TMP_ID )
SVC ON SVC.CLAIM_ID = CLM.CLAIM_ID AND SVC.CD <> ''

SELECT @TMP_ID2

```

The final statement selects the value of the new BIN_ID – in this way, the stored procedure can supply a value back to the automation for the next step in the process.

Returning to our workflow CSVLegacy.xml, we need to add a new variable to the CGateContext object called strBinID that will accept the new output from the stored procedure. Instead of declaring it as a string, we'll just set it as 'object' so that it can be used with the CGateCNNSQLGetScalar activity.

Next, we add the CGateCNNSQLGetScalar activity and copy all values from the CGateCNNSQLExec activity, put in the strBinID reference to the Result field, and then delete the CGateCNNSQLExec activity.

CNNName	"TriggerDB"
CNNSQL	"EXEC USP_PROCESS_CSV " + strEventID + "," + strFileName + ""
DisplayName	CGateCNNSQLGetScalar
Result	strBinID

At this stage, we now have all of the necessary components needed to create a new EDI file. The next few activities will utilize the built-in automation stored procedure for extracting an EDI file from the BIN and then writing it out to the outbound directory:

Drag the CGateTransfer activity to the bottom of the workflow stack, and select Flat < DB Table as Action, SQL Server as DB Type, "TriggerDB" as Data Source, and the following for the Data Table:

```
"EXEC USP_FETCH_CULLED_BIN " + strBinID.ToString()
```

Then, simply drag and drop CGateHKeyFromFlat and CGateXltEncode to the bottom of the stack.

The next activity is another CGateTransfer, this time with the SegPool > File action and a File Path set to this:

```

"C:\chiapas\pipeline\013_CSVLegacy\03_out_edi\" +
Path.GetFileNameWithoutExtension(strFileName) + ".edi"

```

The activity immediately grows a red dot to flag an error. In this case, the .NET function Path.GetFileNameWithoutExtension is part of the System.IO library which is not included by default. Clicking on the Imports tab at the bottom, typing in System.IO in the "Enter or Select Namespace" window and hitting Enter will resolve the error.

The CSV Legacy automation system is now complete.